1993

Image analysis of a pulsating vapour bubble

Steven Brian Harvey
University of Wollongong, sharvey@uow.edu.au

IMAGE ANALYSIS OF A PULSATING VAPOUR BUBBLE

A thesis submitted in fulfilment of the
requirements for the award of the degree

HONOURS MASTER OF ENGINEERING

from

UNIVERSITY OF WOLLONGONG

by

Steven Brian Harvey BE(Hons)

DEPARTMENT OF MECHANICAL ENGINEERING

1993

CONTENTS

ABSTRACT

A technique is presented that enables flow quantities such as the pressure, surface velocity potential and normal surface velocity of a single vapour bubble evolving near a rigid boundary to be determined unobtrusively from its shape history. To achieve this, high-speed ciné film images of the bubble's motion are digitally analysed and the normal surface velocity determined. Application of the boundary integral method enables the potential field, and by use of the Bernoulli equation the pressure field, to be recreated.

An alternative method based on spherical motion of the bubble surface is included for comparison. The fluid is assumed ideal and the flow irrotational. Axisymmetry in the flow field and bubble shape is also assumed.

The relevance of this work lies in its application to the problem of hydraulic cavitation, where vapour bubble collapse and rebound near a solid boundary occur. This is believed to be responsible for the erosion and pitting of hydraulic equipment such as valves, dam spillways and marine propellers. A number of damage mechanisms are believed to exist and these have been numerically modelled in the past. Hence there is a need to correlate these results with experimental data.

Due to the extremely short pulsation period of a bubble (typically 20 msec) quantitative measurements are difficult to make. To this end, high-speed cinématography has been shown to be a useful unobtrusive measurement technique.

ACKNOWLEDGEMENTS

The utilisation of both experimental and theoretical techniques in order to yield useful results has never been an easy task. Any such achievements in this work would not have taken place without the valuable input of others.

So I am deeply indebted to my supervisor Dr Wee-King Soh for his guidance, comments and encouragement during the course of this project. Thanks also are due to Dr Ching-Fu Yu for his efforts in producing some of the best ciné film footage ever seen in this field of engineeering.

For his assistance in the mathematical aspects of this work and constant patience I wish to thank Dr John Best of the Defence Science and Technology Organisation, Melbourne, who has clarified many of the concepts and ideas. Many thanks to Mr Darren Weise, also at the DSTO, for his work in the digital image analysis of the photographic images.

To my wife Gretel, I express my heartfelt appreciation for her understanding and forebearance throughout this project, and her interest in what at times seemed to be an abstract concept. Finally, I thank my parents for their encouragement over the past two years.

The financial support provided by a University of Wollongong Postgraduate Research Award is gratefully acknowledged.

DEDICATION

To the advancement of engineering as a tool for the benefit of mankind.

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

Symbol	Meaning
a	collocation variable
$\mathbf{a}$	vector
b	collocation variable
c	collocation variable
C_d	discharge capacitance
d	length, collocation variable
$\mathbf{e_x}$	Cartesian unit vector - x
$\mathbf{e_y}$	Cartesian unit vector - y
$\mathbf{e_z}$	Cartesian unit vector - z
E	error
E_k	kinetic energy of bubble
E_{spark}	available spark energy
g	magnitude of gravitational acceleration, function
$\mathbf{g}$	gravitational acceleration vector
G	Green's function
h_r	distance from inception point to rigid boundary
H	depth of inception point
i	processed surface counter, collocation counter
j	node counter
J	collocation counter
k	ratio of specific heats, collocation counter
$npts$	no. raw data nodes
n	no. processed nodes-1, vector component, polytropic index
$\mathbf{n}$	normal unit vector
p	collocation variable

$\mathbf{p}$	point lying in the flow domain Ω
p_a	ambient free surface pressure
p_b	computed pressure inside bubble (uniform)
p_c	computed nodal surface pressure
p_{th}	theoretical pressure (adiabatic process)
p_v	vapour pressure of the fluid
p_∞	hydrostatic pressure at inception point
$\mathbf{q}$	point lying on the bubble surface S
r	cylindrical coordinate
$r_A, \ldots, r_D$	spline coefficients
$\mathbf{r}$	flow field geometry
R	mean bubble radius
$\dot{R}$	mean bubble surface radial velocity
$\ddot{R}$	mean bubble surface radial acceleration
R_m	maximum bubble radius
s	distance between nodes
S	bubble surface
t	time, vector component, cumulative distance along surface
$\mathbf{t}$	tangential unit vector
δt	timestep
T_w	water temperature
$\mathbf{u}$	fluid velocity
V	bubble volume
V_d	discharge voltage
x	Cartesian coordinate
y	Cartesian coordinate
z	Cartesian coordinate, cylindrical coordinate
$z_A, \ldots, z_D$	spline coefficients
δ	buoyancy parameter, normal displacement
ϵ	expansion parameter
γ	standoff parameter
μ	function value at bubble centroid
ρ	density of surrounding fluid
ϕ	velocity potential, function

ψ	normal velocity function
Ω	flow domain
$\partial\Omega$	flow domain boundary
Σ	other surfaces bounding Ω
θ	azimuthal angle, orientation of linear elements
ξ	linear parameter, arclength

Chapter 1

INTRODUCTION

The growth and collapse of vapour bubbles in a fluid is a physical phenomenon known as hydraulic cavitation. It is directly responsible for permanent damage to mechanical hydrodynamic elements such as marine propellers, dam spillways and valves. In an engineering sense, this process is one of the few remaining practical problems to be solved.

Cavitation damage by pitting and erosion is thought to be caused by the high pressures induced upon the surface of a rigid boundary when impacted upon by the high-speed fluid jet[1] created during the collapse of a nearby vapour bubble. This thesis makes no comment on the validity of such an hypothesis. Rather, it aims to develop a technique whereby the flow field around an underwater vapour bubble can be reconstructed for further study from experimental photographic data alone.

As with any physical process, the study, analysis and simulation of cavitation is vital to its understanding. Much work has been undertaken to date in the mathematical modelling of the evolution of a single vapour bubble. This approach has its merits, these mainly being the ability of the observer to examine the dynamics of the bubble at a much slower speed and in the absence of experimental noise.

However, in order to assess the validity of computed results there is also a need to reconcile them with experimental results. A convenient source of experimental data is the spark discharge tank, capable of producing a single isolated pulsating vapour bubble near a rigid boundary. Such a device

[1]The *high-speed fluid jet* is believed to play a major role in the process of hydraulic cavitation. Tip speeds of up to 120 m/sec (Lauterborn and Bolle, 1975) have been measured.

is employed in this work. Detailed examination of isolated vapour bubbles yields many measurable flow quantities such as centroidal movement, bubble volume, fluid jet speed and normal surface velocity.

Due to the extremely short pusation period of a bubble (typically 20 msec), measurements must be taken at high speed. An important requirement of any measuring technique is that its sensors do not in any way perturb the quantity being measured. One technique that fulfils this criteria is high-speed cinématography and subsequent digital image analysis which, in addition, has the advantage of enabling the recorded data to be readliy visualised.

To this end, two computational techniques have been developed that analyse the shape history of a bubble as recorded in digitised photographic images. The first is based on the assumption that the bubble evolves in a spherical manner. This is a fair assumption considering that the first pulsation shape histories of typical bubbles are indeed quite spherical. By utilising the radial data derived from such histories the assumed uniform pressure inside the bubble can be estimated.

A more detailed approach is used in the second technique whereby the normal surface velocity of evenly spaced nodes on the bubble's surface is measured from its shape history. The boundary integral method has been implemented numerically to determine the potential field of the experimental bubble from these measurements. Subsequent application of the Bernoulli equation allows the pressure at any point on the surface of the bubble to be determined.

Furthermore, it has been possible to examine the behaviour of a bubble by considering its evolution as a polytropic thermodynamic process. Indeed good, correlation between theory and experiment has been achieved and this has enabled polytropic indices to be calculated and different processes identified.

Chapter 2

EXPERIMENT

This chapter is concerned with the experimental aspects of this work, the results of which are critical to any subsequent numerical analysis scheme. There are many different areas of vapour bubble analysis that can be examined by experiment, such as the formation of the high-speed fluid jet, centroidal movement of the bubble and measurement of its internal pressure. However, due to the very short lifetime of a bubble's evolution, these quantities must be investigated by specialised measurement techniques.

The majority of past works in this field have relied on high-speed cinématography as an unobtrusive method to obtain physical data. In this respect the present work follows similar lines, but develops more fully the combination of digitised shape histories and the boundary integral method. In this sense it is unique and when fully developed may prove to be a very useful experimental technique.

Before progressing onto its detailed description, it is fitting that a review of relevant past and present experimental works should be undertaken.

2.1 Review of experimental developments

During the past three decades numerous experimental investigations into the dynamics of vapour bubbles have been made. Ellis (1956) pioneered a high-speed photographic technique that enabled him to observe the evolution of cavitation bubbles in an acoustic field. Framing rates of up to 1,000,000 per second were achieved by using a rotating-mirror camera. His aim was to correlate existing theories of bubble dynamics with experimental results. He

succeeded in confirming that in accordance with theory, bubble collapse is inherently much less stable than bubble growth.

Naudé and Ellis (1961) proposed a perfect fluid theory for a nonhemispherical bubble which collapses in contact with a solid boundary. High-speed motion pictures (100,000 frames per second) of spark generated bubbles were used to test the theory experimentally. It was postulated that damage to the solid boundary was due to pressures caused by the cavity wall striking it, and these were measured to be as high as 20 MPa.

Shutler and Mesler (1965) investigated the effect of nearby rigid boundaries on the damage capabilites of spark induced bubbles. The evolutions were recorded on ciné film at 8000 frames per second, with a xenon flash tube providing short exposure times. They determined that a bubble collapsing against a solid boundary can form a liquid jet which impinges upon it, but that the jet has little or no damage capability. Rather they believed pressure pulses were the main cause of boundary damage.

Not satisfied with the quality of these works, Benjamin and Ellis (1966) sought to improve the experimental results in order that the damage capabilities of the high-speed fluid jet near a rigid boundary could be observed and measured. This they did and subsequently proposed the hypothesis that the asymmetric collapse and subsequent liquid jet that was observed to thread the bubble and strike a nearby boundary was a prime cause of cavitation damage. This was contrary to the conclusions of Shutler and Mesler (1965).

Gibson (1968) supported Benjamin and Ellis's hypothesis by capturing the evolution of a single bubble near a rigid boundary at 10,000 frames per second using a rotating drum camera. He eliminated bouyancy effects from his observations by using a free-fall cavitation tank. The bubble was generated by electric-spark discharge and the high-speed fluid jet was clearly visible.

Plesset and Chapman (1971) numerically reproduced part of the work of Benjamin and Ellis, but raised doubts as to the soundness of purely experimental methods claiming that they '... *are difficult and give only sketchy results* ...'.

Kling and Hammitt (1972) drew similar conclusions to Benjamin and Ellis (1966) as to the damage capability of the high-speed jet. Their experimental

apparatus generated a vapour cavity in a flowing system. High-speed photography at 1,000,000 frames per second was used to observe its subsequent migration and collapse toward a rigid boundary. Internal bubble pressures of up to 455 atmospheres were quoted. They commented however that while spark-generated bubbles are useful in experimental studies, they differ in important ways from cavitation bubbles.

The interaction of an individual vapour bubble with a neighbouring air bubble was investigated by Smith and Mesler (1972) using high-speed photography (35,000 frames per second). A vapour cavity was generated in water by discharging a capacitor across a pair of tungsten electrodes. They concluded that an air bubble located on a solid boundary was able to protect the surface from damage.

Lauterborn and Bolle (1975) used giant pulses of a Q-switched ruby laser to generate vapour cavities in distilled water. High speed photography at 300,000 frames per second followed by computer image analysis was used to determine bubble-wall velocities. They successfully observed the non-spherical collapse of a bubble, its elongation becoming pronounced in the direction normal to a nearby rigid boundary. Realisation of one of Plesset and Chapman's numerical examples was experimentally achieved. The benefits of laser-produced bubbles and computer image-smoothing techniques were also discussed.

An ultrasonic horn was used by Singer and Harvey (1979) to produce collapsing cavitation bubbles near a stationary specimen of plasticine. Damage to the specimen was observed via the use of still photography and attributed to the impingement of high-velocity microjets. Their conclusions were that whilst the spherical pressure wave cannot be discounted, microjets remain the dominant cause of cavitation damage.

Lauterborn (1982) examined the collapse of a laser-produced bubble and detected a bubble vortex ring after jet formation. A holographic technique was used, producing up to 25,000 holograms per second. However the hologram quality was unsatisfactory, and ordinary high-speed ciné film was taken at 20,000 frames per second for comparison. Laser-induced bubble techniques are also discussed by Lauterborn and Vogel (1984), along with other optical methods in fluid mechanics.

A comparison between the behaviour of spark-generated bubbles in distilled water both with and without drag-reducing polymer additives was made by Chahine (1982) using high-speed photography at 10,000 frames per second. He concluded that the additives weaken the influence that nearby solid boundaries have in affecting the departure of spherocity of the bubble surface and hence the microjet.

Laser-produced cavitation bubbles near a stagnation flow region were examined at 1,000,000 frames per second by Ellis and Starrett (1983). Pressure measurements made at a nearby solid boundary reveal that the shape of the bubble can profoundly affect the mechanical impulse its collapse delivers. Impact pressures of the high-speed jet were recorded as high as 1 GPa and jet velocity estimated to be in the order of 5000 m/sec. They comment however that these results only apply to stagnation flow into a rigid boundary.

The mechanism of impulsive pressure generation from a single bubble collapsing in a steady fluid was studied experimentally by Tomita and Shima (1986). High-speed schlieren photography at 500,000 frames per second enabled the shock waves produced from a spark-induced vapour cavity to be captured and subsequently examined. Pressures induced on a nearby solid boundary were measured using transducers and photoelastic methods. It was noted that the transducer results were only qualitative due to non-uniform spatial sensitivity. It was found that the impulsive pressure was closely related to the behaviour of the high-speed jet.

Vogel et. al. (1989) examined the dynamics of laser-generated cavitation bubbles near a solid boundary. Jet and counterjet formation and vortex-ring development were observed and recorded using high-speed photography at up to 1,000,000 frames per second. The fluid velocity field surrounding the bubble was determined with time-resolved particle image velocimetry. Good agreement with numerical results was shown. The pressure inside the bubble during collapse was measured at up to 2.5 kbar using a hydrophone and optical detection technique. Maximum bubble radii were in the order of 3.5 mm.

Yu and Soh (1992) used high-speed photography at 6000 frames per second to visualise a toroidal vortex upon collapse of a spark-induced bubble near a rigid boundary. Gibson's (1968) bubble generation apparatus was

used to produce larger cavities (17mm maximum radii) than possible with laser techniques. This enabled bouyancy effects to be considered, which were believed to be the cause of an observed microjet away from the rigid boundary.

2.2 Experimental technique

This work employs Gibson's (1968) original experimental apparatus, as modified by Yu and Soh (1992). It enables high quality photographic images of the evolution of a single vapour cavity near a rigid boundary to be obtained. Whilst the apparatus was originally intended to be used for gravity-free tests, in this work the tank remained static so that bouyancy effects could be observed. Figure 2.1 depicts the experiment in schematic form.

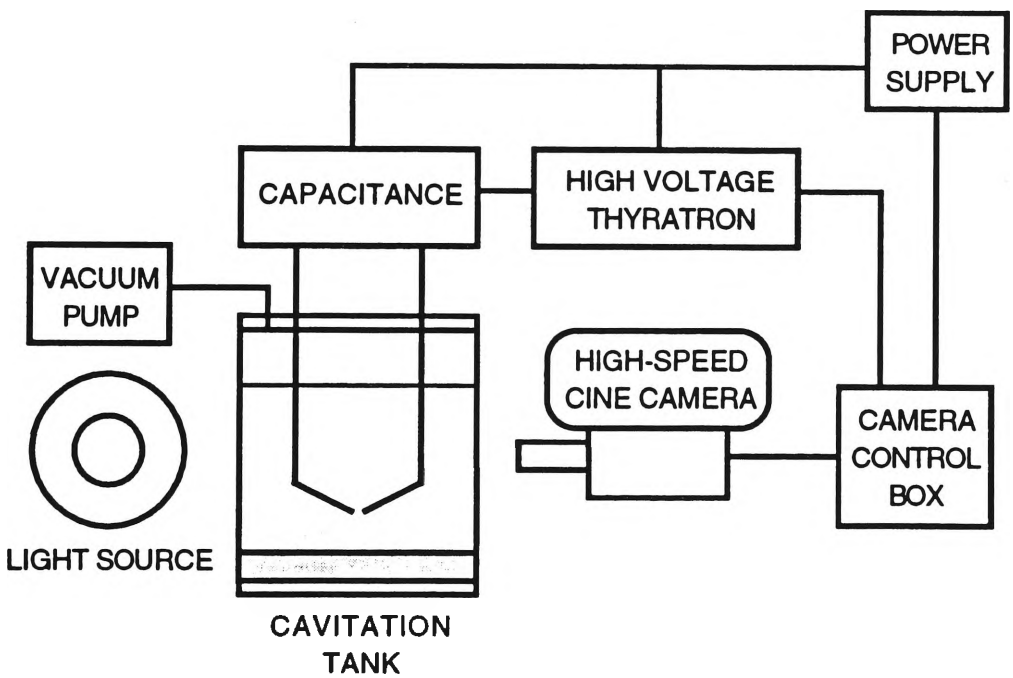

Figure 2.1: Schematic diagram of experimental apparatus

The bubble tank consists of a 260 mm internal-diameter transparent cylinder sealed at both ends. It is formed from 10 mm thick perspex and has an internal depth of 370 mm. Photographic images are obtained through a 25 mm thick flat perspex window in the wall of the tank to reduce optical distortion. A perspex rigid boundary lies at the bottom of the tank, and distilled

water fills the tank to a nominal depth of 250 mm. Two tungsten electrodes are situated above the rigid boundary and the gap between their tips is able to be adjusted as required. Figure 2.2 depicts the experimental parameters used with the bubble tank.

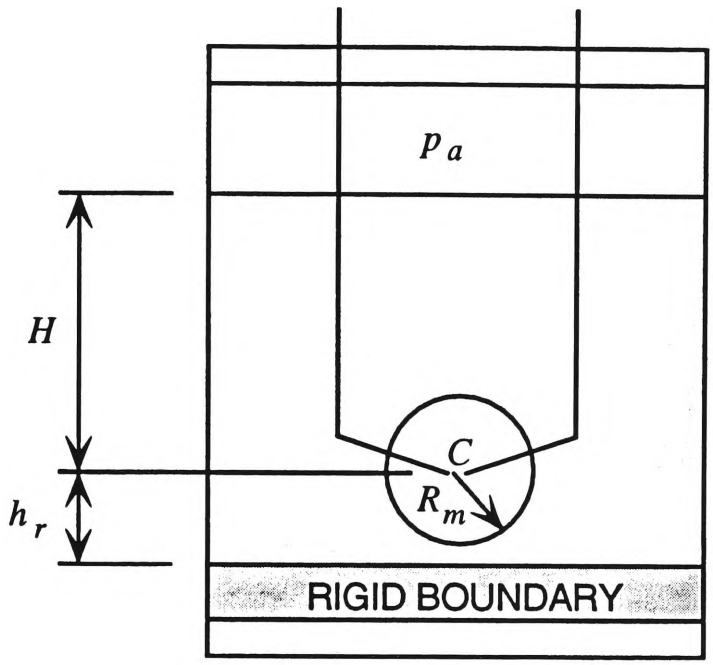

Figure 2.2: Bubble tank experimental parameters

The maximum bubble radius is denoted by R_m. The depth H of inception is the vertical distance from the free surface to the inception point C, and h_r is the distance from C to the rigid boundary. Both H and h_r can be varied by changing the vertical position of the electrodes. The ambient pressure inside the tank at the free surface is denoted by p_a. Thus the hydrostatic pressure p_∞ at the inception point can be expressed as

$$p_\infty = p_a + \rho g H, \tag{2.1}$$

where g is the gravitational acceleration and ρ the fluid density. A vacuum pump connected to the top of the tank maintains p_a at the desired value. This is normally kept as low as possible in order to ensure that a bubble of suitable volume, and hence sufficiently long lifetime, is generated. This factor is crucial since the cinématography in this work uses a relatively low framing rate compared to similar works such as that of Vogel et. al. (1989).

A bubble is generated by applying 10,000 V across the tungsten electrodes which causes a high current pulse to pass between them at C. This results in intense local vapourisation of the water between the electrodes, which in turn creates a vapour bubble in the fluid. The electrode gap is maintained at 0.5 mm in order to ensure high quality electrical discharges. A photograph of the tank is given in Figure 2.3.

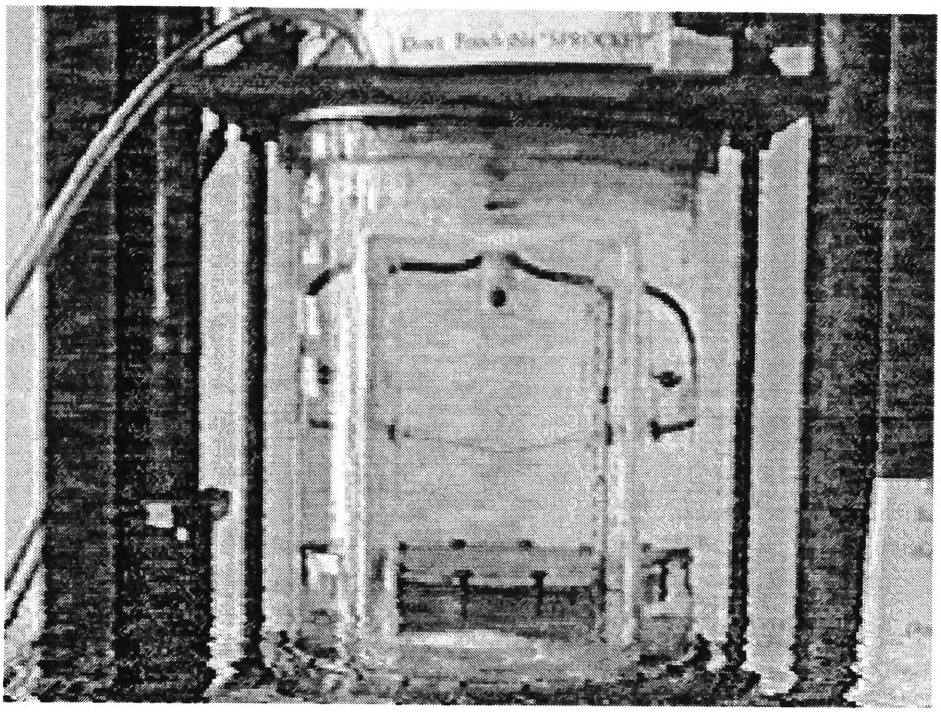

Figure 2.3: Bubble tank

A capacitance of 1 μF is placed across the electrodes, which at 10,000 V, avails approximately 50 J for the vapourisation process. Although Gibson (1972) suggests that only 0.5 to 10 percent of this electrical energy is dissipated in the spark, there is sufficient energy liberated to allow workable experiments to be undertaken.

Dissolved air must be removed from the test water otherwise at low p_∞ small spurious bubbles tend to nucleate on the air particles. These obscure the images of the bubble being photographed by the high-speed camera and hence impede the obtaining of good quality results from from the subsequent image analysis. This degassing is achieved by vibrating and evacuating the bubble tank for up to one hour prior to each experiment.

The evolution of the single vapour bubble is captured on Kodak Tri-x

black-and-white reversal film, using a NAC E-10 high-speed ciné camera. A Micro-Nikkor 55 mm f/2.8 telephoto lens focuses the image into the camera's optical system. The camera and lens combination are shown in Figure 2.4 below.

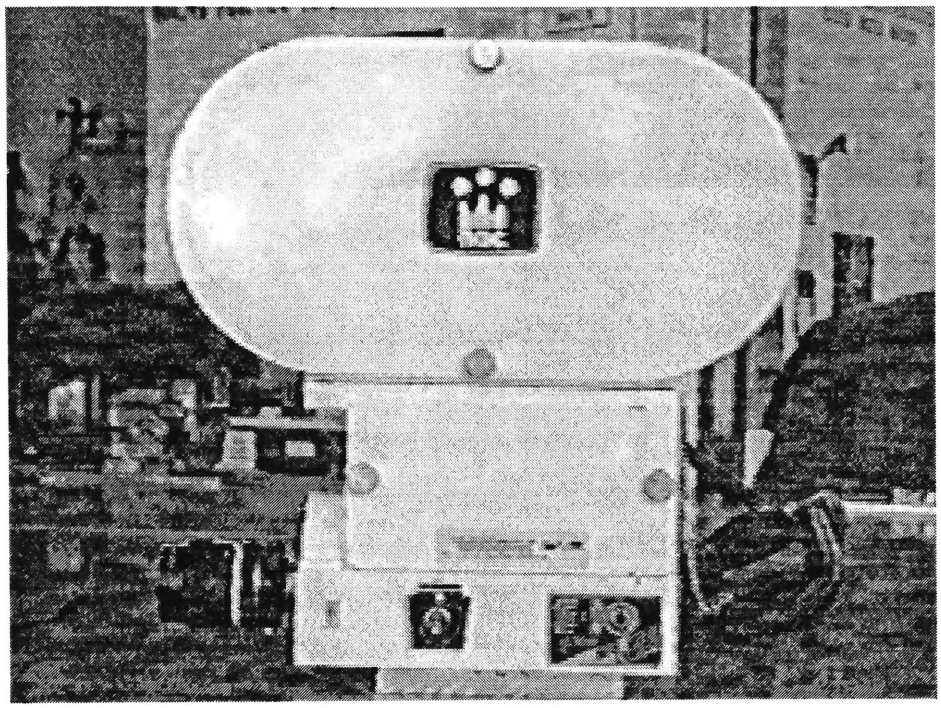

Figure 2.4: High speed ciné camera and lens

The framing rate used is approximately 6000 per second, and a high intensity 2.4 kW ring light provides adequate backlighting. The spark discharge apparatus is synchronised with a signal from the camera, so that when the required framing speed is reached the thyratron is triggered. This in turn discharges the capacitance across the electrodes and forms a vapour bubble.

Film processing is done on site and takes aproximately 30 minutes after which the images can be viewed on a standard 16 mm film projector. Post processing in the form of digital image analysis is done off site using an image analyser which generates Cartesian coordinate files of the bubble surface for every alternate frame in the sequence. A detailed description of this process is given in Appendix A.

It is pertinent at this point to briefly describe two important experimental parameters and their use in predicting gross bubble motion. Due to the lower density of the vapour inside the bubble compared to the surrounding fluid,

a buoyancy force, proportional to the volume of the bubble, acts upon it in the direction opposite to that of the gravity force. Gibson (1967) defines the parameter δ that determines the significance of this buoyancy force as

$$\delta = (\rho g R_m / \Delta p)^{\frac{1}{2}}, \tag{2.2}$$

where $\Delta p = p_\infty - p_v$ and p_v is vapour pressure of the fluid. In a physical sense, δ can be thought of as the ratio of the time taken by the bubble to move R_m under gravity to the time taken to reach the end of the first expansion. In practice, buoyancy forces are only considered significant if R_m is greater than 5 mm.

In addition to buoyancy, a force known as the Bjerknes force will attract the bubble toward the rigid boundary. The buoyancy force may either oppose or support this force depending on the orientation of the boundary. If it lies below the bubble then these forces compete, while the forces will act together when the boundary is above the bubble. An indication of the influence of the Bjerknes force may be gained by means of the standoff parameter γ which is defined by Blake et. al. (1986) as

$$\gamma = h_r / R_m. \tag{2.3}$$

Essentially γ determines the initial location of the bubble, and the Bjerknes force bears some inverse relationship to it.

The Kelvin impulse (Benjamin & Ellis, 1966) corresponds to the apparent inertia of the bubble, and is useful in determining aspects of the gross bubble motion. By utilising the concept of the Kelvin impulse it has been possible to quantify the combined effects of the buoyancy and Bjerknes forces in the form of the $\gamma\delta$ relationship.

For the case of a bubble above a plane rigid boundary, it is postulated (Best, 1991a) that if $\gamma\delta > 0.442$ (corresponding to a positive value of the Kelvin impulse at the time of collapse) then the bubble will migrate away from the rigid boundary. If it is less than 0.442 the bubble will move toward the boundary. So it can be seen that both γ and δ are important parameters in the verification of many numerical results since they allow direct correlation between theory and experiment.

Chapter 3

MATHEMATICAL THEORY OF BUBBLE DYNAMICS

Because the evolution of a vapour cavity is such a high speed phenomena, physical experimentation and recording is difficult. Mathematical analysis overcomes, in a sense, this problem by being able to generate bubble motion at a more measurable pace, albeit theoretical. So, as expected, over the decades since the early pioneering work carried out by Rayleigh (1917), many theoreticians have succeeded in modelling the evolution process. Only those works that are considered to be of relevance will be reviewed here. In this work, the boundary integral method is utilised in what is believed to be a unique way - the reconstruction of the pressure field using photographic data. This chapter will describe the theory underlying this utilisation in preparation for a complete description of the two pressure measurement techniques in Chapter 4.

3.1 History of theoretical developments

The conventional theoretical models of vapour cavities assume the surrounding fluid to be incompressible and inviscid and the flow to be irrotational. Rayleigh (1917) also neglected surface tension in his work on spherically symmetric bubble growth and collapse in an infinite fluid. He derived his well-known radial motion equation

$$R\ddot{R} + \frac{3}{2}\dot{R}^2 + \frac{p_\infty - p(R)}{\rho} = 0 \tag{3.1}$$

from the momentum equation, where R is the bubble radius, p_∞ the liquid pressure a large distance away, $p(R)$ the liquid pressure at the bubble boundary and ρ the liquid density. For the early stages of the evolution of a real bubble in an infinite fluid, (3.1) provides a valid description, the motion being symmetric. However if the bubble shape departs from spherical symmetry or is near a rigid boundary, (3.1) no longer applies and an alternative model must be sought.

Purely numerical solutions to the growth and collapse of a single vapour bubble near a rigid boundary have been leading experimental work in recent years due to the ready availability of powerful computing facilities. Plesset and Chapman's (1971) work is today regarded as a milestone in numerical thinking, and forms the basis of comparison for many other numerical studies. They were able to calculate the complete collapse history of an initially spherical bubble near a rigid boundary, including the formation of a high-speed jet, using a particle-in-cell technique. They drew the now commonly accepted conclusion that cavitation damage is likely to be caused by the impact of the high-speed jet on the solid boundary.

The method of matched asymptotic expansions was used by Chahine and Bovis (1983), where the expansion parameter ϵ is the ratio between the initial spherical bubble radius and its distance from a solid boundary. This lead to a system of differential equations describing non-spherical bubble dynamics which could be solved numerically yielding bubble shape, velocity potential and pressure field. An interesting outcome of this work was that pressures generated on the solid boundary by the collapsing non-spherical bubble were computed to be orders of magnitude higher than for the spherical case.

The modelling of the growth and collapse of axisymmetric cavitation bubbles near a rigid boundary has been made easier in recent years by the use of the boundary integral method. This technique, based on Green's formula, is able to follow the contortions of the cavity shape throughout its evolution and is discussed by Blake et. al. (1986). Whilst during the growth phase a bubble may be approximately spherical, the collapse leads to irregularities in its velocity and pressure field. The boundary integral method successfully computes these quantities and the effect that a nearby rigid boundary has on them.

The present work utilises an inverse form of the boundary integral method of Taib (1985), whereby Green's formula is reduced to a Fredholm integral equation of the second kind and solved for the velocity potential on the bubble surface. This is made possible by the determination of normal surface velocity from the photographic shape history of an evolving bubble near a rigid boundary. Application of the Bernoulli equation enables the pressure field to be computed. Other parameters such as kinetic energy and volume are also able to be calculated.

3.2 Description of the flow field

We assume the fluid to be inviscid, incompressible and the flow irrotational and occupying a domain Ω. Its velocity $\mathbf{u}$ therefore is given by

$$\mathbf{u} = \nabla\phi, \tag{3.2}$$

where the velocity potential ϕ satisfies Laplace's equation

$$\nabla^2\phi = 0. \tag{3.3}$$

Surface tension at the bubble wall is neglected as is any gas flow within the bubble.

The bubble surface S is a subset of the domain boundary $\partial\Omega$ which incorporates both free and rigid boundaries, and Σ consists of all other surfaces that bound Ω. The normal interior to the bubble surface S is denoted by $\mathbf{n}$. A Cartesian coordinate system is specified by the orthogonal vectors $\mathbf{e_x}$, $\mathbf{e_y}$ and $\mathbf{e_z}$ as shown in Figure 3.1 . Gravitational acceleration is defined as $\mathbf{g} = -g\mathbf{e_z}$.

Since there is no flow normal to a rigid boundary, here the velocity potential satisfies

$$\nabla\phi \cdot \mathbf{n} = 0. \tag{3.4}$$

At a free boundary, such as the bubble surface S, the fluid pressure is equal to the pressure external to the fluid.

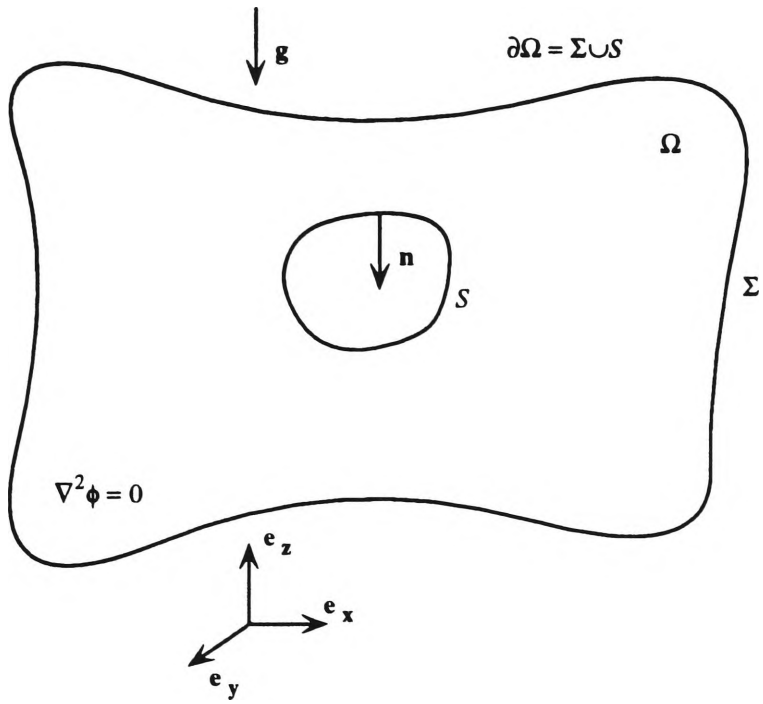

Figure 3.1: The flow field (Best, 1991a)

3.3 The boundary integral method

The motion of an axisymmetric transient cavity in an ideal fluid as defined in Section 3.2 has been successfully computed by Taib (1985). Improvements to the algorithm were made by Kucera (1993) and Best (1991a) to arrive at the numerical technique in its present form. This technique is used in an inverse way in the present work. That is, the motion of a bubble is not computed, but rather the algorithm is reversed to compute its surface velocity potential, given its normal surface velocity as obtained from experimental data.

We now examine the essential elements of the boundary integral method. By applying Green's theorem we can solve Laplace's equation (3.3) in the domain Ω yielding

$$c(\mathbf{p})\phi(\mathbf{p}) = \int_{\partial\Omega} \left(\frac{\partial\phi}{\partial n}(\mathbf{q}) G(\mathbf{p},\mathbf{q}) - \phi(\mathbf{q}) \frac{\partial G}{\partial n}(\mathbf{p},\mathbf{q}) \right) dS(\mathbf{q}), \qquad (3.5)$$

where

$$c(\mathbf{p}) = 4\pi - \int_{\partial\Omega} \frac{\partial G}{\partial n}(\mathbf{p}, \mathbf{q}) dS(\mathbf{q}) \tag{3.6}$$

and

$$c(\mathbf{p}) = \begin{cases} 2\pi & \mathbf{p} \in \partial\Omega, \\ 4\pi & \mathbf{p} \in \Omega \backslash \partial\Omega. \end{cases}$$

The boundary surface $\partial\Omega$ is smooth and $\Omega\backslash\partial\Omega$ denotes its complement in Ω, while the Green's function is given by G. We define the normal derivative at the boundary as $\partial/\partial n \equiv \mathbf{n} \cdot \nabla$. The point $\mathbf{p}$ lies in the flow domain Ω and we define a point $\mathbf{q}$ as lying on the bubble surface S. For motion in an infinite fluid $\partial\Omega \equiv S$ and the Green's function is

$$G_{inf}(\mathbf{p}, \mathbf{q}) = \frac{1}{|\mathbf{p} - \mathbf{q}|}, \tag{3.7}$$

while for motion in the neighbourhood of a rigid boundary

$$G_{rig}(\mathbf{p}, \mathbf{q}) = \frac{1}{|\mathbf{p} - \mathbf{q}|} + \frac{1}{|\mathbf{p} - \mathbf{q}'|}. \tag{3.8}$$

We reflect $\mathbf{q}$ about the rigid boundary to obtain $\mathbf{q}'$ which accounts for the presence of the boundary.

Since the bubble geometry S and $\partial\phi/\partial n$ on S are known from experimental data, (3.5) is a Fredholm integral equation of the second kind. Solving for ϕ on S as a function of time allows the pressure within the experimental bubble to be determined using the Bernoulli equation.

Chapter 4

COMPUTATIONAL TECHNIQUES

In this chapter we examine two techniques for extracting meaningful information from the photographic records of a pulsating vapour bubble near a rigid boundary. The first, rather simplistic approach utilises the Rayleigh radial motion equation (3.1) to compute the bubble pressure and assumes the bubble to remain spherical throughout its lifetime. This assumption is reasonable during the expansion stage of the bubble but breaks down as the bubble collapses asymmetrically.

The second, more realistic approach uses the shape history of the bubble to determine the normal velocity at each of its assigned surface nodes. Upon application of the boundary integral method as described in Section 3.3 the velocity potential at each node is found as a function of time. Using the Bernoulli equation the nodal pressures can then be computed. Other quantities such as volume and kinetic energy are also obtained.

Before proceeding with either method it is fitting that the physical form of the raw experimental data used in this work be considered.

4.1 Experimental data preprocessing

After the experiment has taken place using the procedure described in Section 2.2, the results are in the form of captured negative images on 16 mm ciné film. With a framing rate of 6000 per second and the pulsation period of the bubble approximately being 20 msec, there are perhaps 330 images of the evolving bubble produced per experiment. Of course, this figure will vary but in general it has been found that a sufficiently low p_∞ will yield around

120 images for the first pulsation period.

Once the bubble rebounds and then begins its second collapse, it no longer approximates a spherical geometry but tends to be toroidal in shape. Since the photographic technique can only record two-dimensional images, it becomes difficult to distinguish the bubble shape, because some parts of the bubble will move inside the surface and be obscured from view. Therefore we have confined the bounds of this work to the first pulsation only.

Figure 4.1 shows selected ciné film records of the expansion stage of a typical spark-induced bubble near a rigid boundary. The electrodes are visible, and must be filtered out in the image analysis process. In order to be able to distinguish and digitise the bubble shape the analysis equipment possesses advanced processing capabilities.

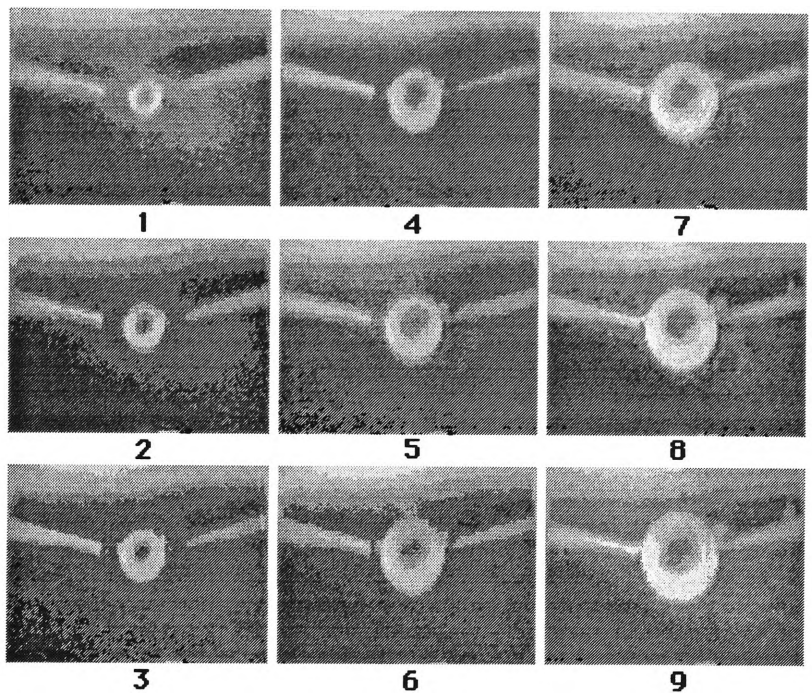

Figure 4.1: Experimental ciné film records, film RB3,$\gamma = 2.041$, $\delta = 0.270$

The bubble surface is found by interactively extracting the image from the background using digitisation followed by greylevel thresholding on a semi-automatic image processing system.

Data is created by assigning nodes to points on the surface of the digitised bubble image at constant angular intervals from the centroid C (x_{cent}, y_{cent}), which is also computed by the analyser. A suitable scale is determined using the known distance between the electrode/tank intersection points.

One set of Cartesian coordinate (x, y) data describing the bubble outline is created by the image analyser for every second frame in the experimental sequence, relative to a known reference point (x_{ref}, y_{ref}), and saved in text form to a data file. The reference point usually lies on the rigid boundary, which for this work lies below the bubble. Figure 4.2 depicts this arrangement in diagrammatic form.

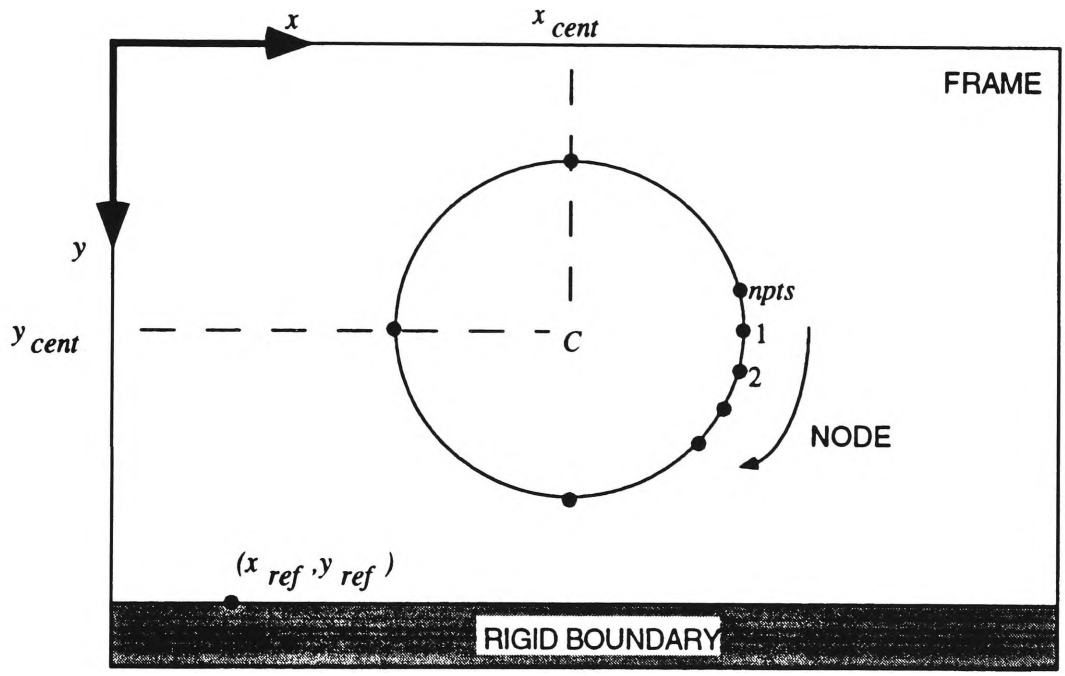

Figure 4.2: Raw data notation after initial digitisation

The number of nodes, *npts*, is arbitrary but a minimum of 36 are typically assigned to ensure that the bubble shapes are adequately defined in two dimensions. The complete image analysis process is described in detail in Appendix A.

A typical coordinate data file would be of the form given in Figure 4.3.

NODE	COORD_X	COORD_Y
1	359	209
2	357	223
3	353	237
4	347	250
5	338	260
6	329	270
7	318	279
.	.	.
.	.	.
.	.	.
29	291	128
30	305	133
31	318	137
32	330	146
33	341	156
34	349	167
35	358	179
36	362	194

Figure 4.3: Typical raw coordinate data file

Coordinate data in the above form must be preprocessed for use in either computational method. The first stage is to translate the origin of each image to (x_{cent}, y_{ref}) which makes it colinear with the vertical axis of the bubble, i.e.,

$$x \Rightarrow x - x_{cent}. \tag{4.1}$$

In applying this algorithm we are assuming that the bubble centroid undergoes no horizontal translation during its lifetime. This is a fair assumption since the dominant bubble forces, i.e., Bjerknes and buoyancy, are exerted in the vertical direction only. All ordinates are translated so that the origin lies on the rigid boundary, that is,

$$y \Rightarrow y - y_{ref}. \tag{4.2}$$

Due to the inherent experimental noise contained in the raw data, some form of surface smoothing is neccesary. This is done by applying the assumption of axisymmetry of the two dimensional image in the vertical axis and using the simple coordinate averaging algorithm

$$x \Rightarrow \frac{x_{right} - x_{left}}{2}, \tag{4.3}$$

where x_{right} and x_{left} represent the right and left abscissae respectively for each ordinate of the bubble surface. Note that the sign of x_{right} will be positive while x_{left} will be negative. This results in (4.3) yielding only positive values for x, and tends to smooth out any major spurious surface noise. Since the bubble is now axisymmetric we need only consider its right hand side, the left hand side being a mirror image. This reduction in the total number of nodes needing analysis speeds subsequent processing time since now $n = npts/2 + 1$.

The final stage of preprocessing involves renumbering the nodes of this half surface and converting them from Cartesian (x, y) to cylindrical (r, z, θ) coordinates as depicted in Figure 4.4.

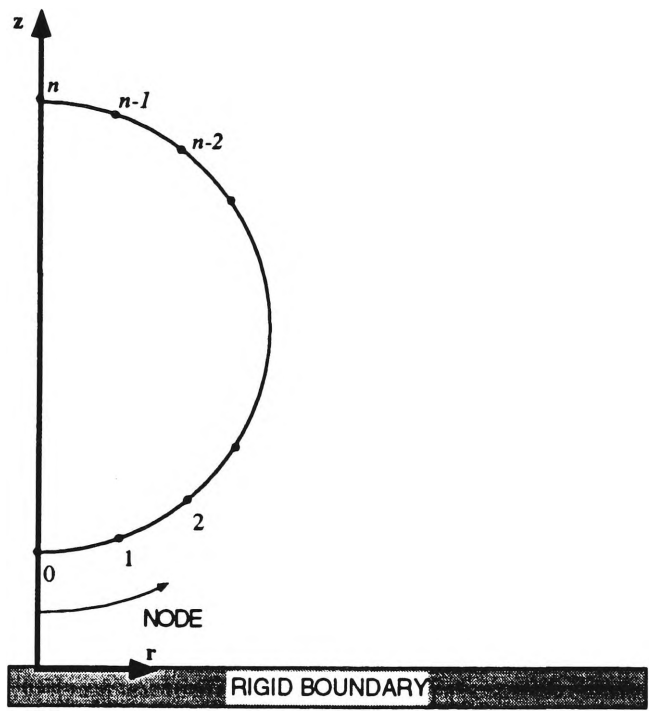

Figure 4.4: Processed data notation

The z axis passes through nodes 0 and n with $z = 0$ occurring at the rigid boundary, and due to the assumption of axisymmetry the azimuthal angle $\theta = 0$. The final form of the typical coordinate files is shown in Figure 4.5.

NODE	R	Z
0	0.0000000	0.0342912
1	0.0026820	0.0344828
2	0.0053640	0.0354406
3	0.0078544	0.0365901
4	0.0100575	0.0383142
5	0.0117816	0.0402299
6	0.0134100	0.0421456
7	0.0145594	0.0446360
8	0.0152299	0.0473180
9	0.0155173	0.0500000
10	0.0159962	0.0528736
11	0.0156130	0.0557472
12	0.0141763	0.0580460
13	0.0119732	0.0601533
14	0.0099617	0.0620690
15	0.0077586	0.0637931
16	0.0052682	0.0645594
17	0.0026820	0.0655173
18	0.0000000	0.0655173

Figure 4.5: Typical processed coordinate data file

Data in this form is used in both computational techniques. Appendix B contains the **FORTRAN** code used to preprocess the raw data files using the method just described.

In what follows certain conventions are adhered to. Firstly, we use the Lagrangian method to describe fluid particle motion. Secondly, all calculations using experimental data are carried out using nondimensionalised values. We scale all distances with respect to the maximum bubble radius R_m, while times are scaled w.r.t. $R_m(\rho/\Delta p)^{\frac{1}{2}}$, where $\Delta p = p_\infty - p_v$. It follows then that pressures are scaled w.r.t. Δp and velocities to $(\Delta p/\rho)^{\frac{1}{2}}$. For quantitative results these parameters are multiplied by the relevant scaling factor after the calculations have been completed.

4.2 Spherical bubble method

The spherical bubble model as presented by Rayleigh (1917) can be rearranged so that given the radial history of the bubble, its pressure as a function of time may be computed. This model assumes that the effects of a nearby boundary are small, that the bubble undergoes only spherical motion and that it has a uniform internal pressure.

It is likely to be valid only during the first pulsation of the bubble, since it begins to break down as the bubble collapses asymmetrically. However it does provide a convenient pressure measurement technique if the shape history of the bubble exhibits spherical motion. A formulation of the method follows.

For a spherical bubble in the neighbourhood of boundaries the potential is

$$\phi \sim -\frac{R^2\dot{R}}{|\mathbf{r}|} - R^2\dot{R}g(\mathbf{r}), \tag{4.4}$$

where $g(\mathbf{r})$ is a function of the flow field geometry $\mathbf{r}$ and R is the radius of the bubble (Best and Blake, 1993). Differentiating (4.4) in time yields

$$\frac{\partial\phi}{\partial t} \sim -\frac{[2R\dot{R}^2 + R^2\ddot{R}]}{|\mathbf{r}|} - [2R\dot{R}^2 + R^2\ddot{R}]g(\mathbf{r}). \tag{4.5}$$

Evaluating this at the surface of the bubble gives

$$\frac{\partial\phi}{\partial t} \sim -2\dot{R}^2 - R\ddot{R} - [2R\dot{R}^2 + R^2\ddot{R}]\mu, \tag{4.6}$$

where

$$\mu = g(\mathbf{r_0}, \mathbf{s}) \tag{4.7}$$

and is small, and $\mathbf{r_0}$ is the position of the bubble centroid at $t = 0$. The geometry of bounding surfaces to the flow domain Ω is characterised by $\mathbf{s}$. For the case of a rigid boundary,

$$\mu = \frac{1}{2|h_r|}, \tag{4.8}$$

and s is considered as h_r. To the required order of accuracy of this work

$$\nabla\phi \sim \frac{R^2\dot{R}}{|\mathbf{r}|^2}, \tag{4.9}$$

and at the bubble surface

$$|\nabla\phi|^2 \sim \dot{R}^2. \tag{4.10}$$

We now introduce the Bernoulli equation in the nondimensional form

$$\frac{\partial\phi}{\partial t} + \frac{1}{2}|\nabla\phi|^2 + p_b + \delta^2 z - 1 = 0, \tag{4.11}$$

where p_b is the time-varying pressure inside the bubble which is uniform and unknown. Now $\delta = O(\mu)$, and since the calculation is done to $O(\mu)$ and μ is small we can neglect $\delta^2 = O(\mu^2)$. Substitution of (4.6) and (4.10) into (4.11) yields

$$R\ddot{R} + \frac{3}{2}\dot{R}^2 + \mu R[R\ddot{R} + 2\dot{R}^2] - p_b + 1 = 0, \tag{4.12}$$

which can be rearranged into the more convenient form

$$p_b = R\ddot{R} + \frac{3}{2}\dot{R}^2 + \mu R[R\ddot{R} + 2\dot{R}^2] + 1. \tag{4.13}$$

The mean radius R_t at time t is found from the bubble volume V_t computed from each raw coordinate file using the technique described in Section 4.3.5, where

$$R_t = \sqrt[3]{\frac{3V_t}{4\pi}}. \tag{4.14}$$

Using central difference approximations we readily find $\dot{R}_t$ and $\ddot{R}_t$ from the time history of R_t by

$$\dot{R}_t = \frac{1}{2\delta t}[R_{t+\delta t} - R_{t-\delta t}], \tag{4.15}$$

and

$$\ddot{R}_t = \frac{1}{[\delta t]^2}[R_{t+\delta t} - 2R_t + R_{t-\delta t}], \tag{4.16}$$

where δt is the scaled timestep between successive coordinate files.

The flow chart for the entire pressure extraction process is given in Figure 4.6 below.

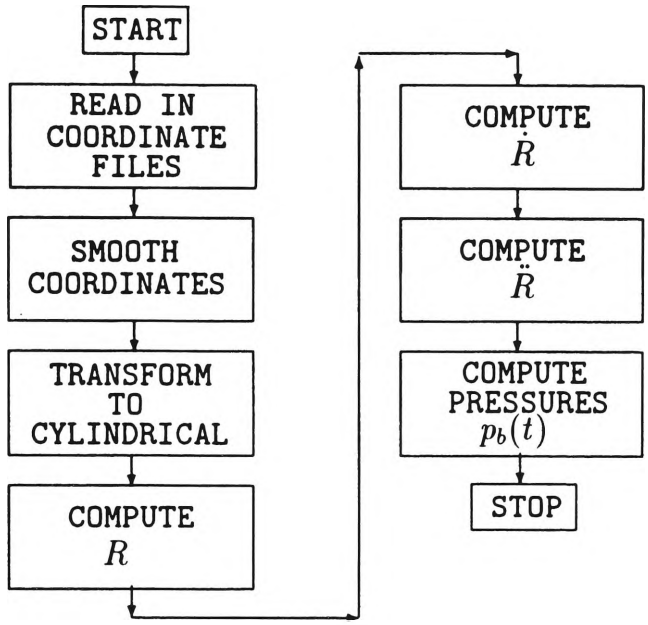

Figure 4.6: Flow chart for spherical bubble method

This technique has been validated with computed coordinate data and the results of this validation are given in Chapter 5. As shall be seen in Chapter 6, the raw experimental data should ideally depict a nearly perfectly spherical bubble, in order to minimise computational error.

4.3 Nodal displacement method

Lauterborn and Bolle (1975) were successful in using computer analysis of photographic images to examine bubble surface movements. They used a digital computer to analyse the photographic history of a vapour bubble and succeeded in calculating the bubble-wall velocities. Although their measurements confirmed numerical results by Plesset and Chapman (1971), they only documented the movements in time of three points on the bubble surface.

Much more information can be gained by assigning evenly spaced nodes over the surface of a bubble. Measurement, in the Lagrangian sense, of their normal velocities as the evolution takes place can then be made from the

recorded shape history. The normal surface velocities $\partial\phi/\partial n$ and the bubble geometry S can then be used to solve (3.5) for the velocity potential ϕ at each node. Furthermore, by applying the Bernoulli equation we can obtain the pressure at each node on the bubble surface.

The nodal displacement method documented here has the advantage of being able to extract meaningful data from non-spherical bubble motion. It has been validated and the these results can be seen in Chapter 5. We shall now examine each step of the technique in detail.

4.3.1 Surface representation

There are two different ways that the surface of the bubble, in its processed nodal form of Figure 4.4, is represented in this work. The first, more simpler method is that of linear interpolation between the nodes. In a physical sense this is the maximum order of interpolation that can be used with any certainty since only the nodes have known positions. We know nothing of the detailed surface between them, other than it must start and end with two consecutive nodes.

For each analysed bubble surface i we denote the linear element j as that which joins nodes $j-1$ and j as in Figure 4.7. At the node j the cylindrical coordinates are (r_j, z_j), the normal fluid velocity with respect to the *interior* normal $(\partial\phi/\partial n)_j \equiv \psi_j$ and the velocity potential is ϕ_j. The distance between nodes is simply the length s_j of the element j or

$$s_j = \left[(r_j - r_{j-1})^2 + (z_j - z_{j-1})^2\right]^{1/2}, \quad j = 1, \ldots, n, \tag{4.17}$$

and the cumulative distance t_j along the bubble surface is given by

$$t_0 = 0, \tag{4.18}$$

and

$$t_j = t_{j-1} + s_j, \quad j = 1, \ldots, n. \tag{4.19}$$

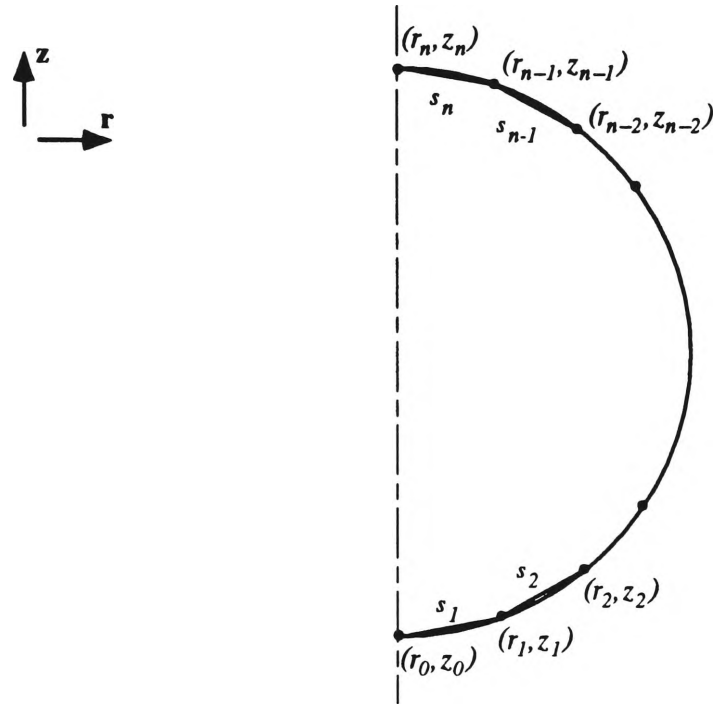

Figure 4.7: Linear element representation of the bubble surface.

The bubble surface and the functions ψ and ϕ are interpolated by using the linear parameter ξ where

$$r(\xi) = r_j + (r_{j+1} - r_j)\frac{(\xi - t_j)}{s_{j+1}}, \quad t_j \leq \xi \leq t_{j+1}, \tag{4.20}$$

$$z(\xi) = z_j + (z_{j+1} - z_j)\frac{(\xi - t_j)}{s_{j+1}}, \quad t_j \leq \xi \leq t_{j+1}, \tag{4.21}$$

$$\psi(\xi) = \psi_j + (\psi_{j+1} - \psi_j)\frac{(\xi - t_j)}{s_{j+1}}, \quad t_j \leq \xi \leq t_{j+1}, \tag{4.22}$$

$$\phi(\xi) = \phi_j + (\phi_{j+1} - \phi_j)\frac{(\xi - t_j)}{s_{j+1}}, \quad t_j \leq \xi \leq t_{j+1}, \tag{4.23}$$

for $j = 0, \ldots, n-1$.

In order to be able to compute the normal surface velocity it is convenient to analyse the surface of the bubble using the normal-tangential $(\mathbf{n}, \mathbf{t})$ coordinate system. In the linearised case we must first specify the orientation of each element. Consider the notation given in Figure 4.8.

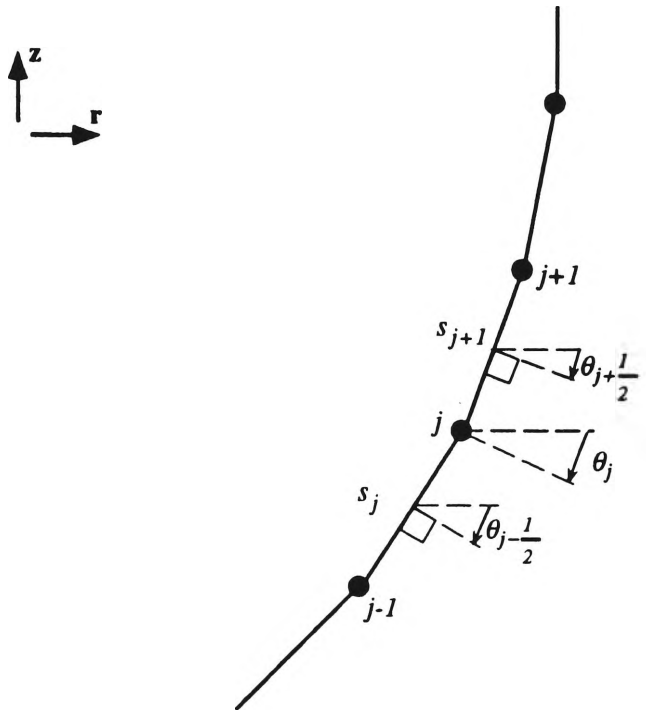

Figure 4.8: Orientation of linear elements.

We introduce $\theta_{j-\frac{1}{2}}$ to denote the orientation of the linear element j. It is given by

$$\theta_{j-\frac{1}{2}} = \arctan\left(\frac{r_{j-1} - r_j}{z_j - z_{j-1}}\right), \quad j = 1, \ldots, n. \tag{4.24}$$

To find the orientation θ_j of the surface at the nodes we use the weighted approximation (Best, 1992)

$$\theta_j = \frac{s_j\theta_{j+\frac{1}{2}} + s_{j+1}\theta_{j-\frac{1}{2}}}{s_j + s_{j+\frac{1}{2}}}, \quad j = 1, \ldots, n-1, \tag{4.25}$$

and at the extremities of the surface

$$\theta_0 = -\frac{\pi}{2}, \tag{4.26}$$

and

$$\theta_n = \frac{\pi}{2}, \tag{4.27}$$

since the bubble is axisymmetric. It follows that the normal **n** and tangential

t unit vectors at each node are given by

$$\mathbf{n}_j = (n_r, n_z)_j = (-\cos\theta_j, -\sin\theta_j), \tag{4.28}$$

$$\mathbf{t}_j = (t_r, t_z)_j = (-\sin\theta_j, \cos\theta_j), \tag{4.29}$$

noting the *inward* orientation of the normal vector in Figure 4.9 below.

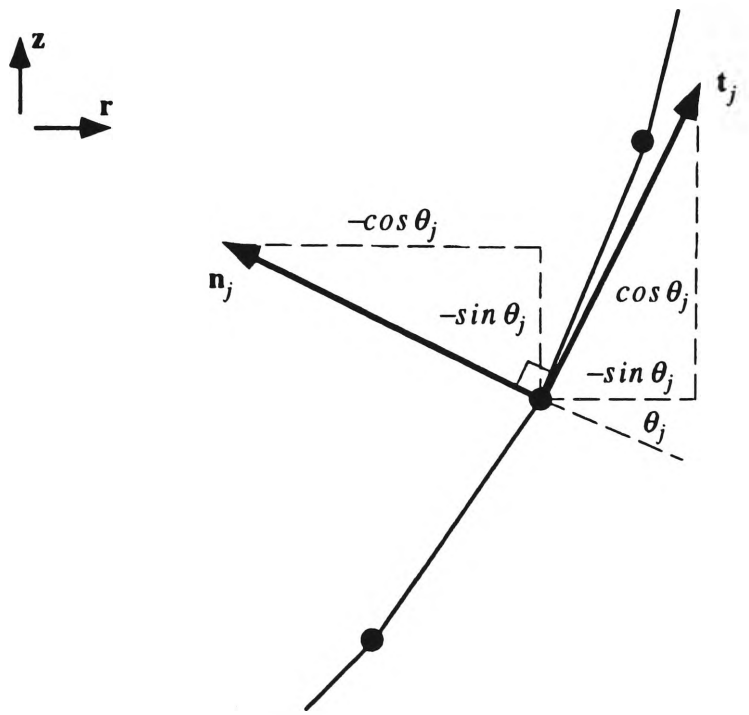

Figure 4.9: Normal and tangential unit vectors at the nodes.

The second, more complex method employs a cubic spline representation of the bubble surface, constrained to pass through the $n+1$ node points of each processed surface. Cubic elements, while offering no more certainty of the detail of the interpolated surface than linear elements, have the advantage of achieving a smooth directional transition at the nodes.

The spline parameter is the numerically found arclength ξ along the bubble surface. If ξ_j is the arclength from node 0 to node j then

$$\delta\xi_j = \xi_j - \xi_{j-1}, \quad j = 1, \ldots, n, \tag{4.30}$$

is the arclength between adjacent nodes, noting that $\xi_0 = 0$. Figure 4.10 illustrates this geometry.

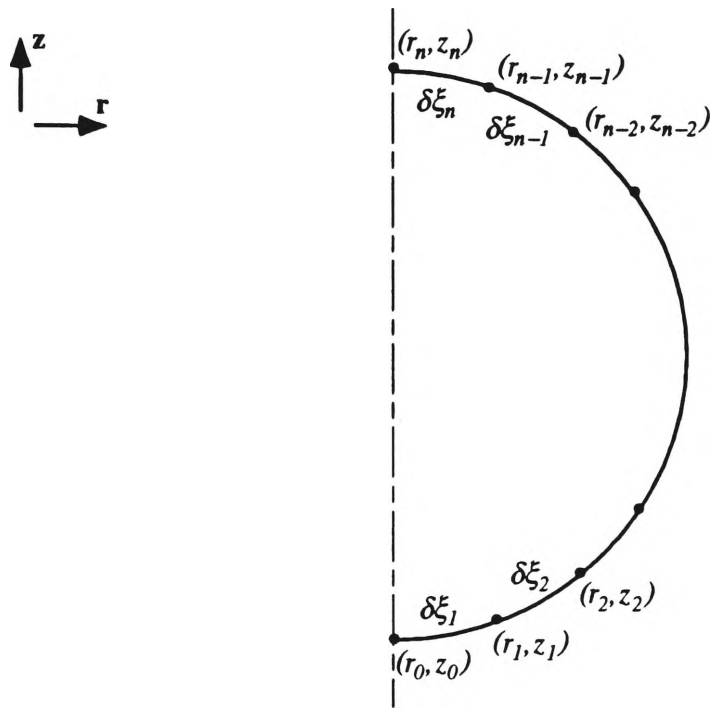

Figure 4.10: Cubic spline representation of the bubble surface.

The spline functions $r(\xi)$ and $z(\xi)$ are expressed as

$$r(\xi) = r_{Aj} + r_{Bj}(\xi - \xi_j) + r_{Cj}(\xi - \xi_j)^2 + r_{Dj}(\xi - \xi_j)^3, \tag{4.31}$$

$$z(\xi) = z_{Aj} + z_{Bj}(\xi - \xi_j) + z_{Cj}(\xi - \xi_j)^2 + z_{Dj}(\xi - \xi_j)^3, \tag{4.32}$$

for

$$\xi_j < \xi < \xi_{j+1}, \quad j = 0, \ldots, n-1, \tag{4.33}$$

and are clamped at the end nodes 0 and n to ensure axisymmetry. The arclength ξ and the spline coefficients are obtained using the iterative improvement technique of Kucera (1993). Clearly at the node points j the interval $\xi - \xi_j = 0$ so that $r_j = r_{Aj}$ and $z_j = z_{Aj}$.

The components of the inward normal **n** and tangential **t** unit vectors at any point on the splined surface are readily found by firstly differentiating (4.31) and (4.32) to give

$$\frac{dr}{d\xi} = r_{Bj} + 2r_{Cj}(\xi - \xi_j) + 3r_{Dj}(\xi - \xi_j)^2, \tag{4.34}$$

and

$$\frac{dz}{d\xi} = z_{Bj} + 2z_{Cj}(\xi - \xi_j) + 3z_{Dj}(\xi - \xi_j)^2, \tag{4.35}$$

whereby

$$\mathbf{n} = (n_r, n_z) = \left(-\frac{dz}{d\xi}, \frac{dr}{d\xi}\right) \tag{4.36}$$

and

$$\mathbf{t} = (t_r, t_z) = \left(\frac{dr}{d\xi}, \frac{dz}{d\xi}\right). \tag{4.37}$$

The orientation of these components is given in Figure 4.11.

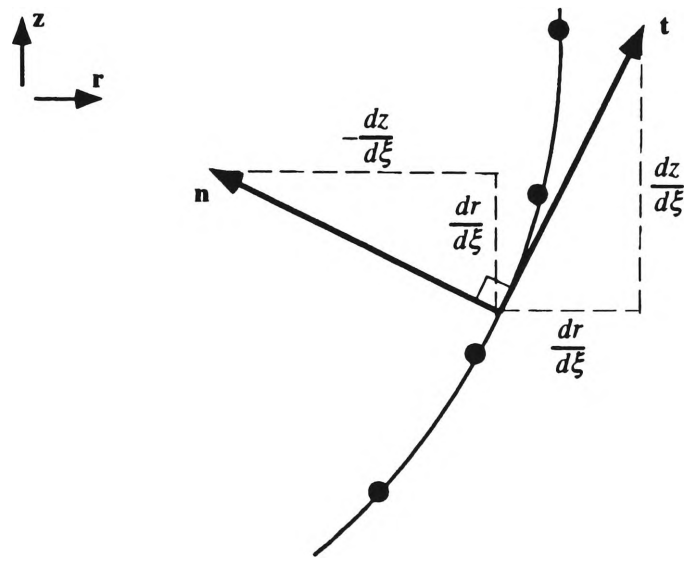

Figure 4.11: Normal and tangential unit vectors

At the nodes,

$$\mathbf{n}_j = (-z_B, r_B)_j, \tag{4.38}$$

and

$$\mathbf{t}_j = (r_B, z_B)_j. \tag{4.39}$$

In comparing linear and cubic spline methods we can say that both are valid in representing the surface of the bubble and indeed each has certain advantages over the other. However, as will be explained in Chapter 6 it was found that the linear representation lacked directional coherence at the

node points. This meant that it was difficult to determine the normal surface velocities and therefore the bubble pressures there. Hence the cubic spline representation was used exclusively throughout this work in order to obtain acceptable results.

4.3.2 Technique for computing nodal $\partial\phi/\partial n$

Critical to the proper behaviour of the nodal displacement method is the correct determination of the nodal normal surface velocities $\partial\phi/\partial n$, which are used in the boundary integral method. The method described here lends itself to experimental bubble dynamics, being able to process the coordinate data of a non-spherical bubble evolution in a robust manner, yielding the normal surface velocity at each node.

Consider the notation of Figure 4.12:

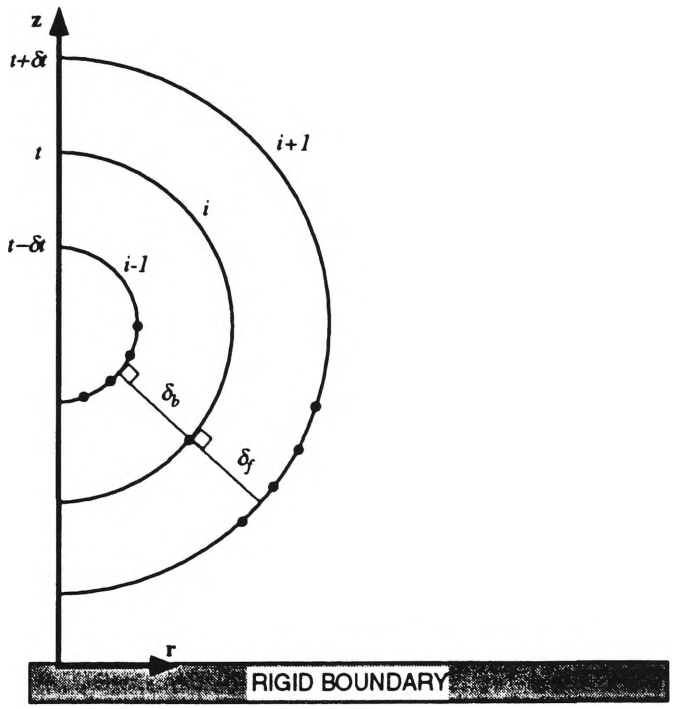

Figure 4.12: Notation used for computing nodal $\partial\phi/\partial n$

We define the distance propagated by a node on the ith surface in the direction normal to the $(i-1)$th surface from the $(i-1)$th surface to the ith surface as the backward normal displacement δ_b. The forward normal displacement of the node from the ith surface to the $(i+1)$th surface in the

direction normal to the ith surface is denoted by δ_f. We evaluate the normal surface velocity $\partial\phi/\partial n$ by combining these two displacements in the central difference equation

$$\frac{\partial\phi}{\partial n} = -\left(\frac{\delta_b + \delta_f}{2\delta t}\right), \tag{4.40}$$

where δt is the scaled timestep between each successive coordinate file. This calculation is done nodewise ($i = 2, \ldots, n-1$) over the first pulsation of the bubble. If the velocities are to be given in physical units then the scaled values of (4.40) must be multiplied by the scale factor $(\Delta p/\rho)^{\frac{1}{2}}$.

Note that $\partial\phi/\partial n$ is a signed variable and is calculated with respect to the *interior* normal. Thus the method is useful in that it can detect, for instance, when the $(i+1)$th surface lies within the ith surface, as is the case for a collapsing bubble.

We now consider, in more detail, the calculation of the normal displacements δ_b and δ_f. The cubic spline surface representation is chosen to illustrate the method, although linear elements have also been used, but with limited success.

Backward normal displacement

Consider two consecutive bubble surfaces each having of format given in Figure 4.4. We adopt the usual notation, designating the former surface $i-1$, occurring at time $t - \delta t$, and the latter i occurring at time t. It is required that the backward normal displacement δ_b of the node P on the surface i along the direction of the normal $\mathbf{n}$ from the surface $i-1$ be determined.

We define a vector $\mathbf{a}$ from a sliding point Q $(r(\xi), z(\xi))$ on the surface $i-1$ to the node P (r_P, z_P) as

$$\mathbf{a} = ([r(\xi) - r_P], [z(\xi) - z_P]), \tag{4.41}$$

and this is shown in Figure 4.13. The length d of the vector $\mathbf{a}$ and hence the distance between P and Q is given by

$$d = \left[[r(\xi) - r_P]^2 + [z(\xi) - z_P]^2\right]^{\frac{1}{2}}. \tag{4.42}$$

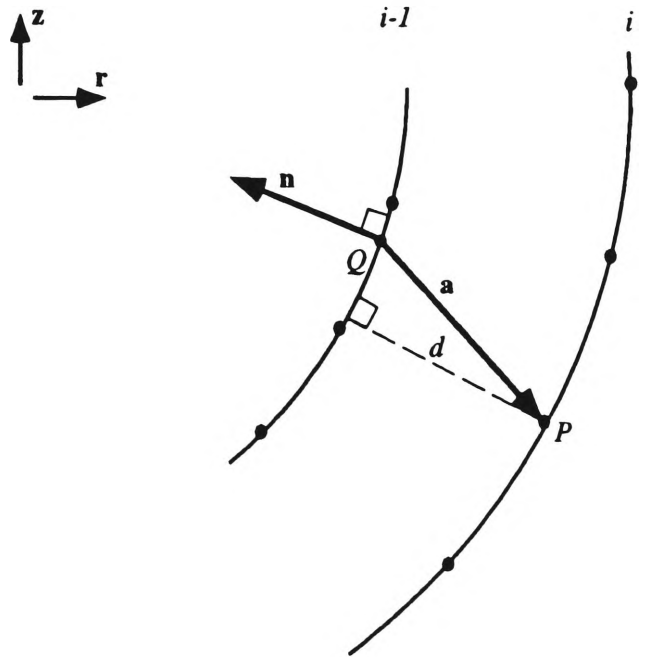

Figure 4.13: Notation for backward normal displacements

If Q is moved along the spline until d^2 is at its minimum or

$$\frac{d(d^2)}{d\xi} = 2\left[r(\xi) - r_P\right]\frac{d}{d\xi}r(\xi) + 2\left[z(\xi) - z_P\right]\frac{d}{d\xi}z(\xi) = 0, \tag{4.43}$$

that is, when

$$\left[r(\xi) - r_P\right]\frac{d}{d\xi}r(\xi) + \left[z(\xi) - z_P\right]\frac{d}{d\xi}z(\xi) = 0, \tag{4.44}$$

we can solve (4.44) for ξ using the secant method. The backward normal displacement δ_b is then found by computing d in (4.42), noting that it is a signed value since the bubble can be either expanding or contracting. The scalar product

$$\mathbf{n}\cdot\mathbf{a} = n_r a_r + n_z a_z = -\frac{d}{d\xi}z(\xi)[r(\xi) - r_P] + \frac{d}{d\xi}r(\xi)[z(\xi) - z_P] \tag{4.45}$$

is then used in determining its sign, so that if $\mathbf{n}\cdot\mathbf{a} < 0$, then $\delta_b = -d$ and we have an expanding bubble, while if $\mathbf{n}\cdot\mathbf{a} > 0$, $\delta_b = d$ and the bubble is contracting.

Forward normal displacement

We again consider two consecutive bubble surfaces, this time designating the former surface i, occurring at time t and the latter $i+1$ surface as occurring at time $t+\delta t$. It is required that the forward normal diplacement δ_f of the node P on the surface i along the direction of the normal $\mathbf{n}$ from the surface i be determined.

A vector $\mathbf{a}$ from a sliding point Q $(r(\xi), z(\xi))$ on the surface $i+1$ to the node P is defined in (4.41), and with the tangential unit vector $\mathbf{t}$ at P whose components are given in (4.37) is shown in Figure 4.14.

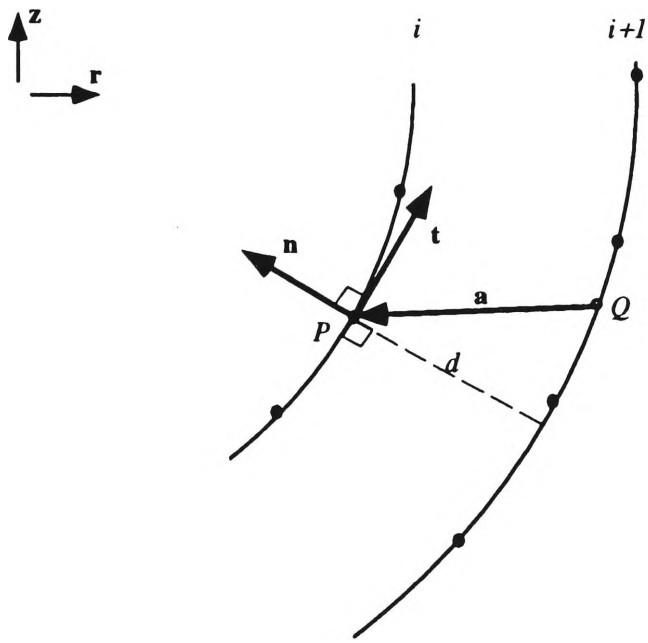

Figure 4.14: Notation for forward normal displacements

Once again the scalar product is used, this time in locating the point Q such that when

$$\mathbf{t} \cdot \mathbf{a} = 0, \tag{4.46}$$

that is, when

$$t_r a_r + t_z a_z = r_B[r(\xi) - r_P] + z_B[z(\xi) - z_P] = 0 \tag{4.47}$$

is solved for ξ using the secant method, the vectors are orthogonal and Q is located.

The distance d is then computed using (4.42), with (4.45) used to determine its sign. If $\mathbf{n} \cdot \mathbf{a} > 0$, then $\delta_f = -d$ and we have an expanding bubble, while if $\mathbf{n} \cdot \mathbf{a} < 0$, then $\delta_f = d$ and the bubble is contracting.

4.3.3 Finding ϕ by the boundary integral method

The boundary integral method which is incorporated into the present work (Best, 1991b) uses the nodal normal surface velocities ($\partial\phi/\partial n$) calculated using the techniques in Section 4.3.2 and shape S of the bubble surface as described by the coordinate files to numerically solve (3.5). This becomes a Fredholm integral equation of the second kind and its solution yields the nodal velocity potentials ϕ.

Consider the linearised bubble surface at time t as depicted in Figure 4.7, where the nodal coordinates are given by (4.20) and (4.21). Initially the distance between each node s_j is given by (4.17) but is subsequently improved upon by use of Kucera's (1993) iterative improvement technique.

Linear interpolation over the bubble surface is now used to represent the normal velocity function $\psi(\xi)$ and velocity potential function $\phi(\xi)$ so that

$$\psi(\xi) = \psi_j + (\psi_{j+1} - \psi_j)\frac{(\xi - t_j)}{s_{j+1}}, \quad t_j \leq \xi \leq t_{j+1}, \tag{4.48}$$

and

$$\phi(\xi) = \phi_j + (\phi_{j+1} - \phi_j)\frac{(\xi - t_j)}{s_{j+1}}, \quad t_j \leq \xi \leq t_{j+1}, \tag{4.49}$$

for $j = 0, \ldots, n-1$. In order to obtain a solution for ϕ_j a collocation technique is used, where (3.5) holds at the nodes which are the collocation points. Upon substitution of (4.20), (4.21), (4.48) and (4.49) into (3.5) we obtain

$$c_j\phi_j = a_j - \sum_{J=1}^{n-1}[b_{jJ}\phi_J + d_{jJ+1}\phi_{J+1}], \quad j = 1, \ldots, n, \tag{4.50}$$

where

$$a_j = \sum_{J=1}^{n-1}\int_0^{2\pi}\int_{t_J}^{t_{J+1}} \psi(\xi)G_j(\xi,\theta)r(\xi)d\xi d\theta, \tag{4.51}$$

$$b_{jJ} = \int_0^{2\pi}\int_{t_J}^{t_{J+1}} \left(\frac{t_{J+1} - \xi}{s_{J+1}}\right)\frac{\partial G_j}{\partial n}(\xi,\theta)r(\xi)d\xi d\theta, \tag{4.52}$$

$$c_j = 4\pi - \sum_{J=1}^{n-1} \int_0^{2\pi} \int_{t_J}^{t_{J+1}} \frac{\partial G_j}{\partial n}(\xi,\theta) r(\xi) d\xi d\theta, \tag{4.53}$$

$$d_{jJ+1} = \int_0^{2\pi} \int_{t_J}^{t_{J+1}} \left(\frac{\xi - t_J}{s_{J+1}} \right) \frac{\partial G_j}{\partial n}(\xi,\theta) r(\xi) d\xi d\theta. \tag{4.54}$$

The coefficient b_{jk} is defined for $k = 1,\ldots,n-1$, d_{jk} for $k = 2,\ldots,n$ and we define $b_{jn} = 0$ and $d_{j1} = 0$. In terms of the cylindrical coordinates the surface area element is

$$dS = r(\xi) d\xi d\theta, \tag{4.55}$$

and the rigid boundary Green's function, from equation (3.8), is

$$G_j(\xi,\theta) = \frac{1}{|\mathbf{p}_j - \mathbf{q}(\xi,\theta)|} + \frac{1}{|\mathbf{p}_j - \mathbf{q}'(\xi,\theta)|}, \tag{4.56}$$

where

$$\mathbf{p}_j \equiv (r_j, z_j, 0), \tag{4.57}$$

the azimuthal angle set to zero due to the axisymmetric assumption, and

$$\mathbf{q}(\xi,\theta) \equiv (r(\xi), z(\xi), \theta). \tag{4.58}$$

In order to evaluate a_j, c_j, b_{jk} and d_{jk}, the integration over θ is performed analytically, which yields expressions involving elliptic integrals of the first and second kind. The integration over the arclength ξ is performed using Gauss-Legendre quadrature formulæ. For singular integrands, the logarithmic singularity is subtracted and the integration is completed using an appropriate quadrature scheme (Taib, 1985).

The potential ϕ_j is then found by solving by direct inversion the linear system

$$\sum_{J=1}^{n} p_{jJ} \phi_J = a_j \tag{4.59}$$

where

$$p_{jJ} = b_{jJ} + d_{jJ} + c_j \delta_{jJ}, \quad j,J = 0,\ldots,n. \tag{4.60}$$

4.3.4 Technique for computing the nodal pressures

In order to compute the nodal pressures we utilise the Bernoulli equation in the nondimensionalised form

$$\frac{\partial \phi}{\partial t} + \frac{1}{2}|\nabla \phi|^2 + p_c + \delta^2 z - 1 = 0, \tag{4.61}$$

applied to each node on the bubble surface and where p_c is the computed pressure at the node. The buoyancy parameter squared, δ^2, is assumed experimentally small and is set to zero. Since the partial derivative $\partial\phi/\partial t$ is difficult to measure experimentally, we substitute

$$\frac{\partial \phi}{\partial t} = \frac{D\phi}{Dt} - |\nabla \phi|^2 \tag{4.62}$$

into (4.61) and rearrange yielding

$$p_c = \frac{1}{2}|\nabla \phi|^2 - \frac{D\phi}{Dt} + 1, \tag{4.63}$$

where the total derivative $D\phi/Dt$ can be readily found from the shape history of the bubble using the techniques that are developed in this section. It is essential this derivative be accurately calculated in order to successfully compute the nodal pressures at the surface of the bubble.

The quantity $|\nabla\phi|$ is simply the magnitude of the velocity vector **u** and its square can be expressed as

$$|\nabla \phi|^2 = \left(\frac{\partial \phi}{\partial n}\right)^2 + \left(\frac{\partial \phi}{\partial \xi}\right)^2, \tag{4.64}$$

which when substituted into (4.63) yields the nodal pressure equation

$$p_c = \frac{1}{2}\left[\left(\frac{\partial \phi}{\partial n}\right)^2 + \left(\frac{\partial \phi}{\partial \xi}\right)^2\right] - \frac{D\phi}{Dt} + 1, \tag{4.65}$$

from which the computed results are derived. The entire nodal displacement method can be summarised in Figure 4.15.

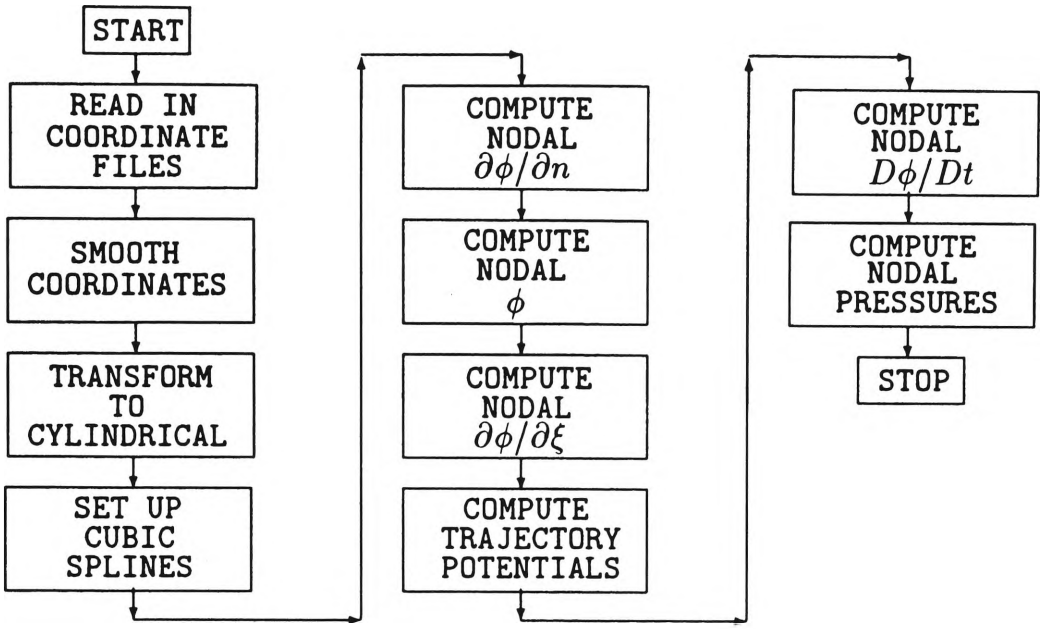

Figure 4.15: Flow chart for nodal displacement method

In order to obtain the pressures in actual physical units the scaled values from (4.65) are multiplied by the parameter Δp.

A preliminary stage in the calculation of $|\nabla\phi|$ is the determination of the tangential surface velocity $\partial\phi/\partial\xi$ at each node. To do this we fit a local quadratic function to the $(j-1)$th, jth and $(j+1)$th nodes and parameterise this with respect to the arclength between nodes $\delta\xi_j$. This is then differentiated to obtain

$$\left(\frac{\partial\phi}{\partial\xi}\right)_j = \frac{\delta\xi_j^2\phi_{j+1} - (\delta\xi_j^2 - \delta\xi_{j+1}^2)\phi_j - \delta\xi_{j+1}^2\phi_{j-1}}{\delta\xi_j\delta\xi_{j+1}(\delta\xi_j + \delta\xi_{j+1})}, \tag{4.66}$$

for $j = 1, \ldots, n-1$. Due to the axisymmetric assumption,

$$\left(\frac{\partial\phi}{\partial\xi}\right)_0 = 0, \tag{4.67}$$

and

$$\left(\frac{\partial\phi}{\partial\xi}\right)_n = 0. \tag{4.68}$$

We choose to represent $\partial\phi/\partial\xi$ between the nodes by the linear function

$$\frac{\partial\phi}{\partial\xi}(\xi) = \left(\frac{\partial\phi}{\partial\xi}\right)_j + \left[\left(\frac{\partial\phi}{\partial\xi}\right)_{j+1} - \left(\frac{\partial\phi}{\partial\xi}\right)_j\right] \frac{(\xi - t_j)}{s_{j+1}}, \quad t_j \le \xi \le t_{j+1}, \quad (4.69)$$

for $j = 0, \ldots, n-1$, where ξ is the linear arclength parameter.

Consider now the bubble surfaces and trajectories depicted in Figure 4.16.

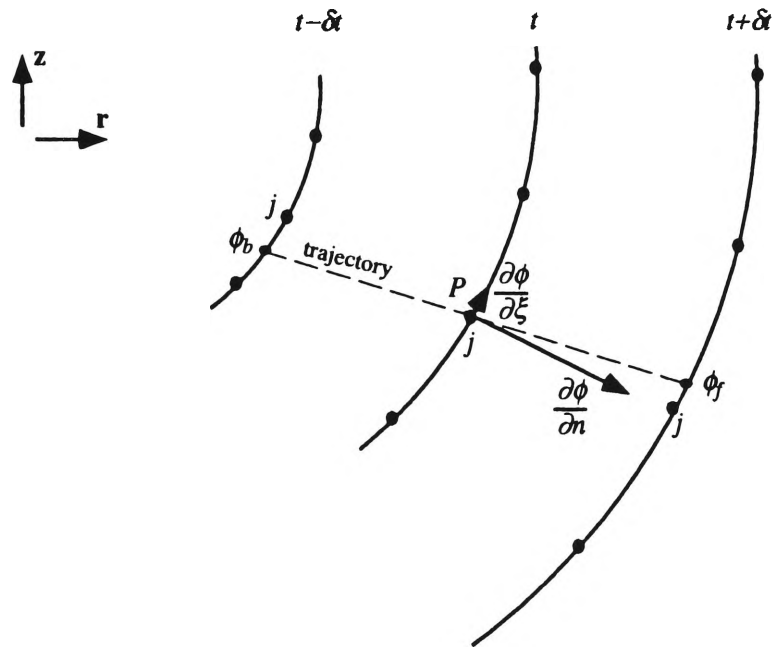

Figure 4.16: Trajectory calculation

We compute the total derivative $D\phi/Dt$ of the node P using the central difference formula

$$\frac{D\phi}{Dt} = \frac{\phi_f - \phi_b}{2\delta t}, \quad (4.70)$$

where ϕ_f is its potential looking *forward* to time $t + \delta t$ and ϕ_b is its potential looking *backward* to time $t - \delta t$, following the trajectory described by the velocity vector

$$\mathbf{u} = \nabla\phi = \frac{\partial\phi}{\partial n}\mathbf{n} + \frac{\partial\phi}{\partial\xi}\mathbf{t}. \quad (4.71)$$

In the case of purely spherical motion, $\partial\phi/\partial\xi = 0$ since the trajectory is in the radial direction. Therefore for each node j in this simplified case we would have

$$\phi_f = \phi_{t+\delta t}, \quad (4.72)$$

and

$$\phi_b = \phi_{t-\delta t}, \tag{4.73}$$

the trajectory intersections coinciding exactly with the node points at these times. However in reality there is always a small tangential velocity component $\partial\phi/\partial\xi$ and so (4.72) and (4.73) are only approximations. In this section we describe vector techniques used to find ϕ_f and ϕ_b for each node on the bubble surface by first computing their probable trajectories. This enables (4.70) to be used to calculate nodal $D\phi/Dt$.

Forward trajectory

Consider two consecutive bubble surfaces, i occurring at time t and $i+1$ occurring at time $t+\delta t$. Note that in the case of a contracting bubble, the $(i+1)$th surface will lie within the ith surface.

We propose that the node P on the ith surface will follow a trajectory *forward* in time as defined by the velocity vector $\mathbf{u}$ in (4.71), to arrive at point W at time $t+\delta t$. Figure 4.17 shows this arrangement.

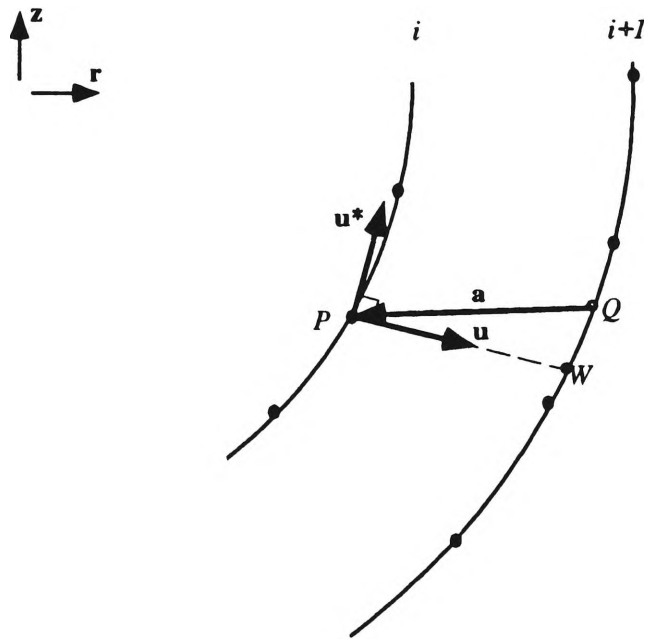

Figure 4.17: Forward trajectory of the node P

A vector $\mathbf{a}$ from a sliding point Q $(r(\xi), z(\xi))$ to the node P is defined

in (4.41). The velocity vector $\mathbf{u}$ at the node P can be expanded into its cylindrical components by substituting (4.36) and (4.37) into (4.71) thus

$$\begin{aligned}
\mathbf{u} &= \frac{\partial\phi}{\partial n}\mathbf{n} + \frac{\partial\phi}{\partial\xi}\mathbf{t} && (4.74)\\
&= \frac{\partial\phi}{\partial n}(n_r, n_z) + \frac{\partial\phi}{\partial\xi}(t_r, t_z) && (4.75)\\
&= \left(\left[\frac{\partial\phi}{\partial n}n_r + \frac{\partial\phi}{\partial\xi}t_r\right], \left[\frac{\partial\phi}{\partial n}n_z + \frac{\partial\phi}{\partial\xi}t_z\right]\right) && (4.76)\\
&= \left(\left[\frac{\partial\phi}{\partial n}\left(-\frac{dz}{d\xi}\right) + \frac{\partial\phi}{\partial\xi}\frac{dr}{d\xi}\right], \left[\frac{\partial\phi}{\partial n}\frac{dr}{d\xi} + \frac{\partial\phi}{\partial\xi}\frac{dz}{d\xi}\right]\right). && (4.77)
\end{aligned}$$

The normal $\mathbf{u}^*$ to $\mathbf{u}$ is then simply

$$\begin{aligned}
\mathbf{u}^* &= -\frac{1}{\mathbf{u}} && (4.78)\\
&= \left(-\left[\frac{\partial\phi}{\partial n}\frac{dr}{d\xi} + \frac{\partial\phi}{\partial\xi}\frac{dz}{d\xi}\right], \left[\frac{\partial\phi}{\partial\xi}\frac{dr}{d\xi} - \frac{\partial\phi}{\partial n}\frac{dz}{d\xi}\right]\right). && (4.79)
\end{aligned}$$

The scalar product between $\mathbf{u}^*$ and $\mathbf{a}$ is

$$\mathbf{u}^*\cdot\mathbf{a} = -\left[\frac{\partial\phi}{\partial n}\frac{dr}{d\xi} + \frac{\partial\phi}{\partial\xi}\frac{dz}{d\xi}\right][r(\xi) - r_P] + \left[\frac{\partial\phi}{\partial\xi}\frac{dr}{d\xi} - \frac{\partial\phi}{\partial n}\frac{dz}{d\xi}\right][z(\xi) - z_P], \quad (4.80)$$

and this is used to locate the trajectory intersection point W by sliding Q along the surface until

$$\mathbf{u}^*\cdot\mathbf{a} = 0. \quad (4.81)$$

This is accomplished by solving (4.81) for ξ using the secant method. The potential ϕ_W at W is then calculated from the linear function for ϕ in (4.23) and this yields

$$\phi_f = \phi_W. \quad (4.82)$$

Backward trajectory

Once again we consider two consecutive bubble surfaces, i occurring at time t and $i-1$ occurring at time $t - \delta t$. Clearly in the case of a contracting bubble, the ith surface will lie within the $(i-1)$th surface.

On the $(i-1)$th surface we represent the normal surface velocity $\partial\phi/\partial n$ by the linear function in (4.22) and the tangential velocity $\partial\phi/\partial\xi$ by the linear

function given in (4.69). We assume that the node at P on the ith surface will have followed the trajectory defined by $\mathbf{u}$ in (4.69) from its position Q $(r(\xi), z(\xi))$ at time $t - \delta t$ to W at time t, as depicted in Figure 4.18 below.

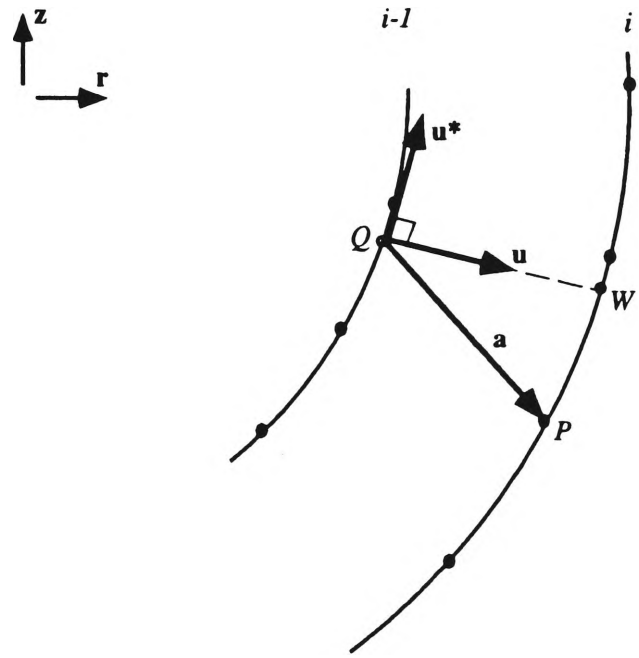

Figure 4.18: Backward trajectory of the node P

We retain the definition of the vector $\mathbf{a}$ given in (4.41) but redefine the velocity vector $\mathbf{u}$ to apply at any point on the $(i-1)$th surface thus

$$\mathbf{u} = \frac{\partial \phi}{\partial n}(\xi)\mathbf{n} + \frac{\partial \phi}{\partial \xi}(\xi)\mathbf{t} \tag{4.83}$$

$$= \frac{\partial \phi}{\partial n}(\xi)[n_r(\xi), n_z(\xi)] + \frac{\partial \phi}{\partial \xi}(\xi)[t_r(\xi), t_z(\xi)] \tag{4.84}$$

$$= \left(\left[\frac{\partial \phi}{\partial n}(\xi) n_r(\xi) + \frac{\partial \phi}{\partial \xi}(\xi) t_r(\xi) \right], \left[\frac{\partial \phi}{\partial n}(\xi) n_z(\xi) + \frac{\partial \phi}{\partial \xi}(\xi) t_z(\xi) \right] \right) \tag{4.85}$$

$$= \left(\left[\frac{\partial \phi}{\partial n}(\xi) \left(-\frac{dz}{d\xi}(\xi) \right) + \frac{\partial \phi}{\partial \xi}(\xi) \frac{dr}{d\xi}(\xi) \right], \left[\frac{\partial \phi}{\partial n}(\xi) \frac{dr}{d\xi}(\xi) + \frac{\partial \phi}{\partial \xi}(\xi) \frac{dz}{d\xi}(\xi) \right] \right), \tag{4.86}$$

and hence the normal $\mathbf{u}^*$ to $\mathbf{u}$ becomes

$$\mathbf{u}^* = -\frac{1}{\mathbf{u}} \tag{4.87}$$

$$= \left(- \left[\frac{\partial \phi}{\partial n}(\xi)\frac{dr}{d\xi}(\xi) + \frac{\partial \phi}{\partial \xi}(\xi)\frac{dz}{d\xi}(\xi) \right], \left[\frac{\partial \phi}{\partial \xi}(\xi)\frac{dr}{d\xi}(\xi) - \frac{\partial \phi}{\partial n}(\xi)\frac{dz}{d\xi}(\xi) \right] \right). \tag{4.88}$$

The scalar product between $\mathbf{u}^*$ and $\mathbf{a}$

$$\mathbf{u}^* \cdot \mathbf{a} = - \left[\frac{\partial \phi}{\partial n}(\xi)\frac{dr}{d\xi}(\xi) + \frac{\partial \phi}{\partial \xi}(\xi)\frac{dz}{d\xi}(\xi) \right] [r(\xi) - r_P] + \left[\frac{\partial \phi}{\partial \xi}(\xi)\frac{dr}{d\xi}(\xi) - \frac{\partial \phi}{\partial n}(\xi)\frac{dz}{d\xi}(\xi) \right] [z(\xi) - z_P], \tag{4.89}$$

is used to locate the trajectory intersection point W by sliding Q along the $(i-1)$th surface until W is coincident with P, i.e.,

$$\mathbf{u}^* \cdot \mathbf{a} = 0. \tag{4.90}$$

This is done by solving (4.90) for ξ using the bisection method, which defines the point Q. The potential ϕ_Q is then calculated from the linear function for ϕ in (4.23) yielding

$$\phi_b = \phi_Q. \tag{4.91}$$

4.3.5 Computing the volume and kinetic energy

When the processed bubble surface is represented by either a cubic spline or a set of linear elements, the volume of the whole bubble at any time in its evolution may be readily calculated.

Consider an infinitesimal slice of thickness dz and radius r which has a constant azimuthal angle θ due to the axisymmetric assumption. We can write the volume dV of the slice as

$$dV = \pi r^2 dz, \tag{4.92}$$

noting that

$$dz = \left(\frac{dz}{d\xi} \right) d\xi. \tag{4.93}$$

By combining (4.92) and (4.93) and integrating nodewise over the bubble surface S we obtain the total volume V of the bubble,

$$V = \int_S dV = \int_{\xi_0}^{\xi_n} \pi r^2 \left(\frac{dz}{d\xi} \right) d\xi. \tag{4.94}$$

The conversion of this scaled value to physical units is accomplished by multiplying by the factor R_m^3.

Any expansion or contraction of a bubble is associated with a change in kinetic energy. At a given point in time this kinetic energy E_k can be written as

$$E_k = \frac{1}{2}\rho \int_V |\nabla\phi|^2 dV \tag{4.95}$$

$$= \frac{1}{2}\rho \int_S \phi \frac{\partial\phi}{\partial n} dS, \tag{4.96}$$

and since

$$dS = 2\pi r d\xi, \tag{4.97}$$

the energy equation (4.96) becomes

$$E_k = \pi\rho \int_S \phi \frac{\partial\phi}{\partial n} r d\xi. \tag{4.98}$$

In order to obtain the kinetic energy in phisical units, the scaled values calculated from (4.98) must be multiplied by the factor $\frac{1}{2}R_m^3\Delta p$.

This completes the description of the techniques used to compute the pressure of a vapour bubble from its shape history. Appendix C contains a listing of the **FORTRAN** code that applies these techniques.

Chapter 5

VALIDATION

Before any computational technique can be utilised, it must first undergo some form of validation comparing its results with known results. This is true for both the spherical bubble and nodal displacement methods of computing the pressure on the bubble surface.

For this work, the known results are generated from bubble dynamics simulation code based on the mathematical model described in Chapter 3. The model (Best 1991a) employs an elementary description of the bubble contents, assuming them to be a combination of liquid vapour and non-condensible products which are ideal and undergo adiabatic expansions and contractions whilst exerting a uniform pressure over the surface of the bubble. The time varying theoretical pressure p_{th} within the bubble can be expessed as a function of its volume V at a given time t by

$$p_{th} = p_v + p_0[V_0/V]^k, \tag{5.1}$$

where p_v is the vapour pressure of the fluid which is constant throughout the evolution. The subscript 0 denotes the initial quantities while k is the ratio of specific heats.

To generate test data we assume that the non-condensible bubble contents consist of an ideal diatomic gas only with 5 degrees of freedom (air). The effects of buoyancy are neglected, as is heat exchange with the surrounding fluid. Furthermore, the presence of liquid vapour is neglected, resulting in the polytropic process described by

$$p_{th} = p_0[V_0/V]^k, \tag{5.2}$$

where for air $k = 1.4$. By stepping the boundary integral code based on this model through time the complete evolution of a bubble can be generated, both in an infinite fluid or near a rigid boundary.

Note that test coordinate data is generated using variable timesteps but extracted at regular time intervals to yield the desired number of coordinate files. Each has the same format as the preprocessed experimental files and contains the cylindrical coordinates of evenly spaced nodes $(0, \ldots, n)$ on the half-surface of a theoretical bubble, as in Figure 4.4. The number of nodes, which equals $n + 1$, may be set arbitrarily, where n is the index of the final node. Figure 5.1 depicts the pulsation of such a generated evolution in the vicinity of a rigid boundary seen at the bottom of the figure. It was produced using the graphic display program in Appendix D.

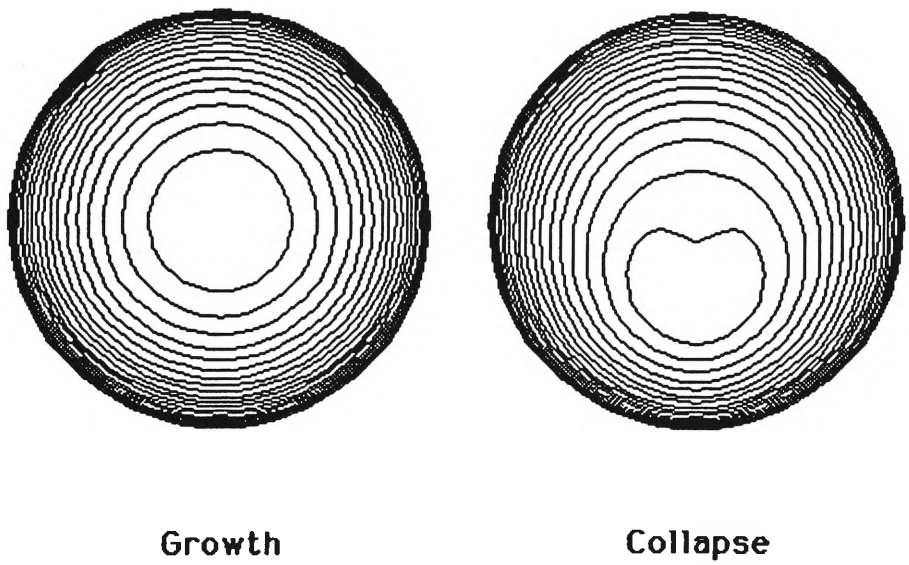

Figure 5.1: Theoretical evolution of a vapour bubble

The validation data also contains the theoretical nodal $\partial\phi/\partial n$, nodal ϕ and bubble pressures in the nondimensional form described in Section 4.1. This enables comparisons to be made with values computed from the validation coordinate data using the two computational techniques described in this work. Data thus produced will be free from any experimental noise. Therefore any errors detected in the validation process can be attributed

solely to the numerical processes involved.

Selection of a suitable theoretical timestep δt and number of nodes is important because the evolution should closely resemble an experimental one in nondimensional duration, number of coordinate files and number of nodes. By selecting $\delta t = 0.025$ and $n = 18$ approximately 84 files are generated for the first pulsation, which is equivalent to a typical experiment where 70 files of 19 nodes are produced. Initial conditions for the theoretical model are pressure $p_0 = 100.0$, inception coordinate $h_r = -2.0$ and the initial bubble radius $R_0 = 0.1651$. The cubic spline surface representation is used for the computed results. If more accuracy is required for the validation process, the timestep may be reduced and/or the number of nodes increased.

5.1 Spherical bubble method

Validation of this method involves comparing the theoretical and computed pressures which are derived from the same set of generated coordinate data. The theoretical pressures p_{th} are generated by (5.2) while the computed pressures p_b are calculated from (4.13). A comparison using generated data with $\delta t = 0.025$ and $n = 18$ is given in Figure 5.2.

Note the close correlation between the results. Clearly for the smooth generated coordinate data depicted in Figure 5.1 the spherical bubble method behaves well in computing the bubble pressure.

However, toward the end of the evolution at approximately time $t = 1.75$ there is a notable divergence between the two sets of results. This is due to the nonspherical geometry the bubble surface begins to acquire at this time and highlights the limitations of the method, which assumes constant radial velocity $\dot{R}$.

In order to gauge the difference between the two sets of values the relative error in the computed pressure E_{rel} is used, where

$$E_{rel} = \frac{|p_{th} - p_b|}{p_{th}}. \tag{5.3}$$

This error is depicted graphically in Figure 5.3.

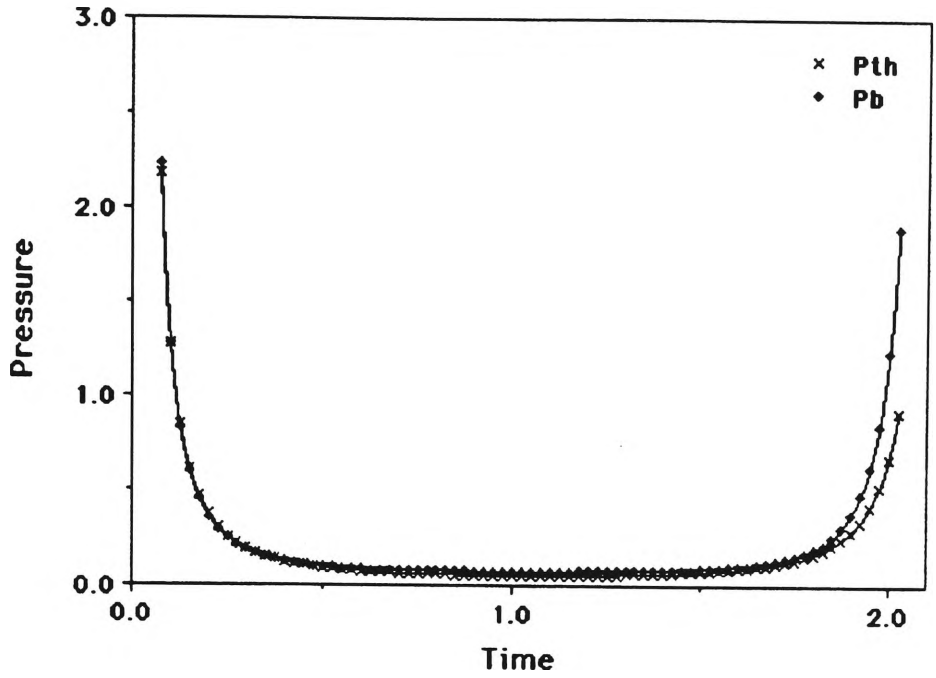

Figure 5.2: Theor. p_{th} and comp. p_b pressures ($\delta t = 0.025$, 19 nodes)

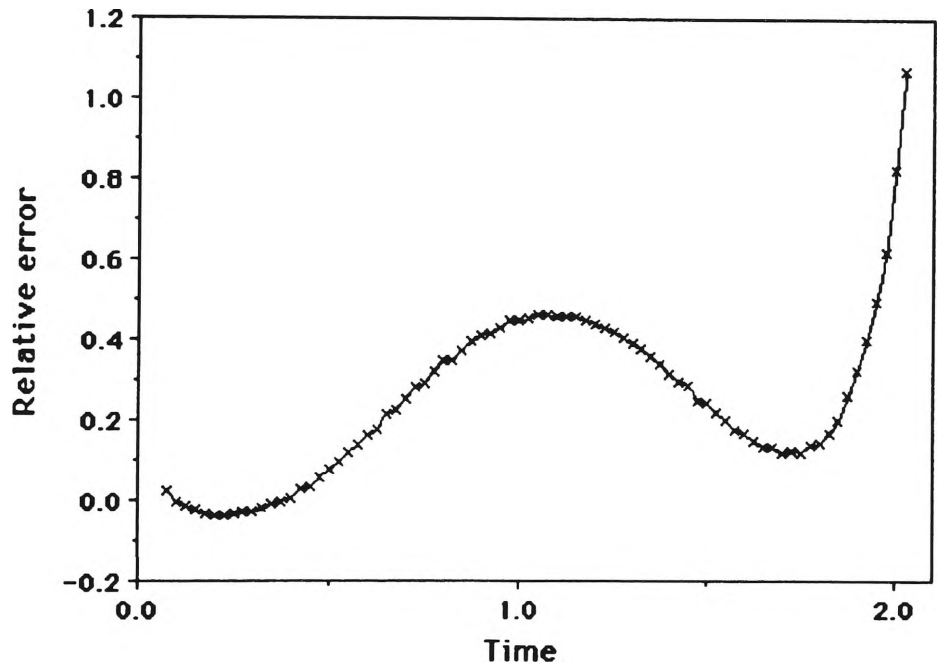

Figure 5.3: Rel. error in computed pressure p_b ($\delta t = 0.025$, 19 nodes)

The start of the first collapse of the bubble occurs near $t = 1.0$, where we note that the spherical bubble method appears to be least accurate. This is misleading however since as the denominator of (5.3) becomes smaller, as is the case near $t = 1.0$, the relative error E_{rel} increases and is unrepresentative of the true error there.

Perhaps a better indication of the accuracy of the spherical bubble method lies in the use of the absolute error E_{abs} which is given by

$$E_{abs} = p_{th} - p_b, \tag{5.4}$$

and can be seen in Figure 5.4.

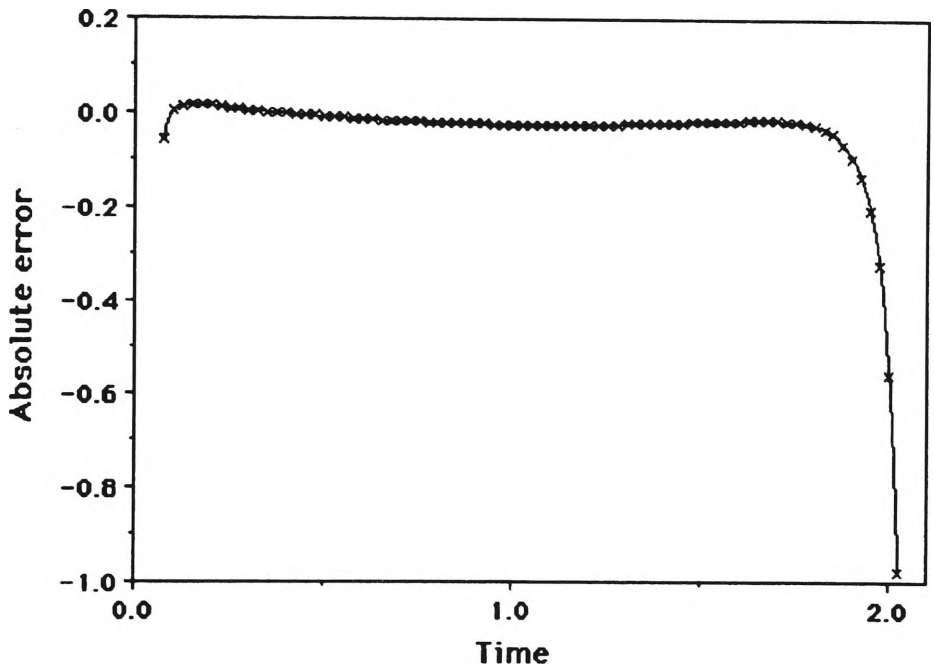

Figure 5.4: Abs. error in computed pressure p_b ($\delta t = 0.025$, 19 nodes)

The divergence after time $t = 1.75$ seen in Figure 5.2 is clearly visible from the large absolute error there. It is evident, however, that the method is indeed well behaved during the majority of the evolution, particularly around the start of collapse at time $t = 1.0$.

Let us now examine the effect of increasing the number of nodes to 37, that is, $n = 36$. Retaining the timestep as $\delta t = 0.025$, the pressures are compared in Figure 5.5. Whilst the theoretical and computed curves coincide for the majority of the evolution, there is still a divergence after time $t = 1.75$. However it is much less than for Figure 5.2 where only 19 nodes are used.

In fact the computed pressure p_b is lower than the theoretical pressure p_{th} in this region. A better idea of this difference can be gauged from the absolute error plot given in Figure 5.6.

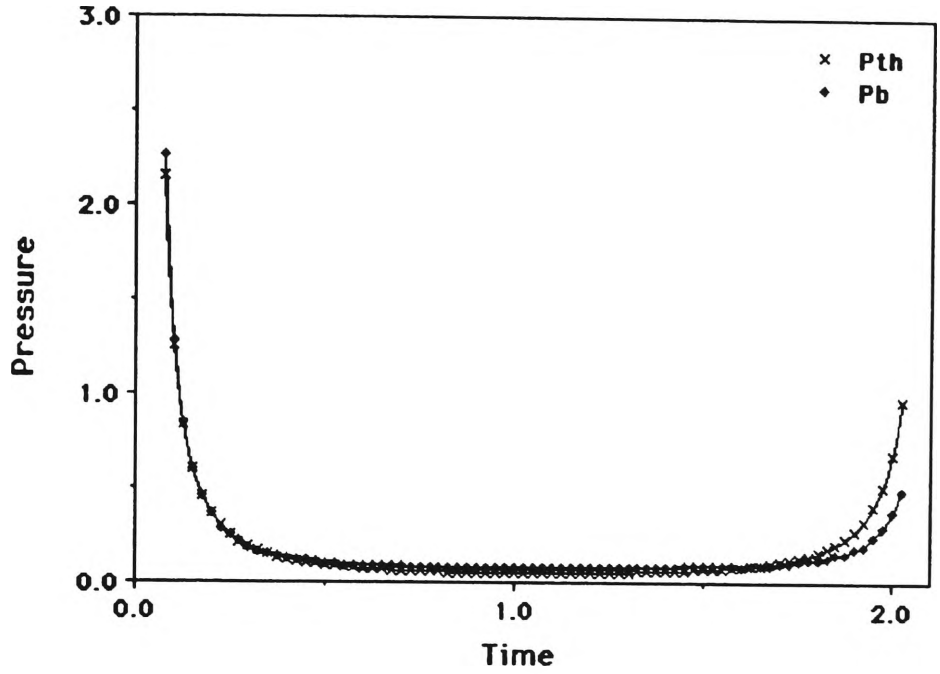

Figure 5.5: Theor. p_{th} and comp. p_b pressures ($\delta t = 0.025$, 37 nodes)

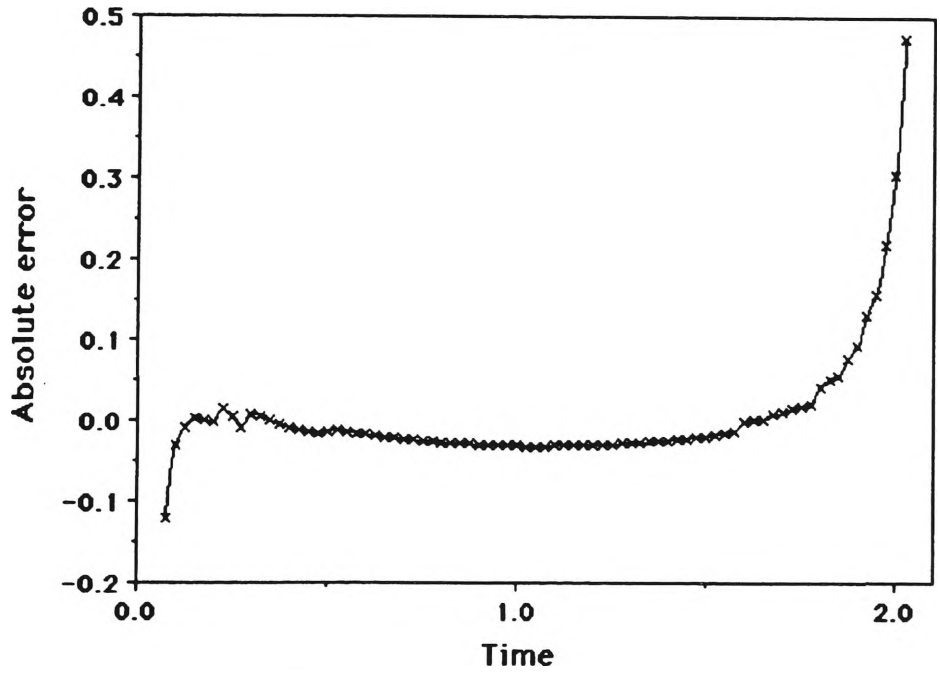

Figure 5.6: Abs. error in computed pressure p_b ($\delta t = 0.025$, 37 nodes)

Clearly the increase in the number of nodes has improved the absolute error, which is about half of that shown in Figure 5.4 with 19 nodes. If the timestep is now reduced by one-half to $\delta t = 0.0125$ there appears to be very little gain in accuracy, as indicated by Figure 5.7.

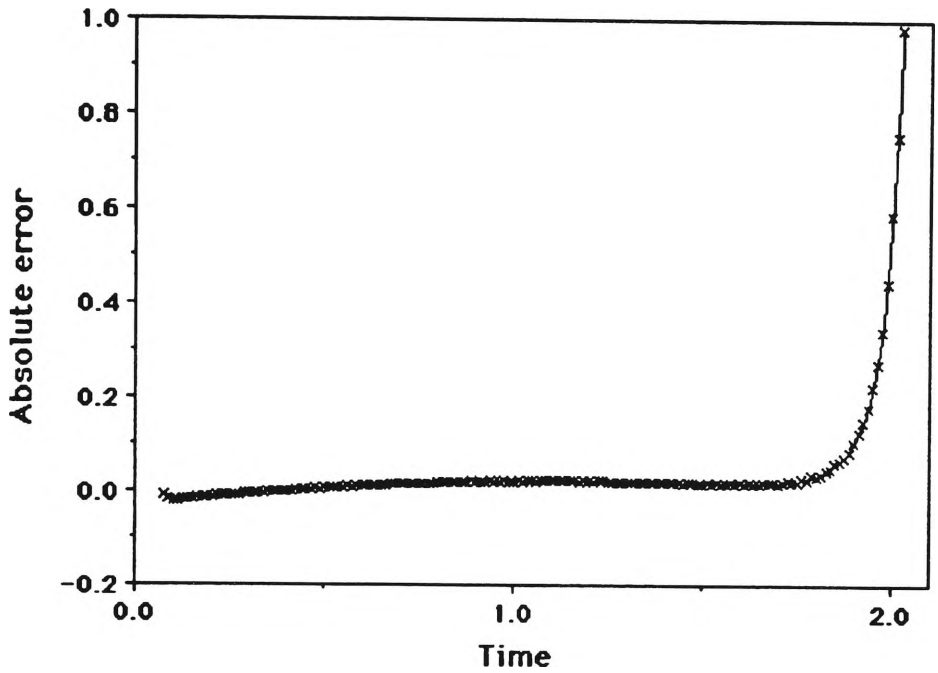

Figure 5.7: Abs. error in computed pressure p_b ($\delta t = 0.0125$, 37 nodes)

A comparison between this graph and Figure 5.4 reveals that in fact the accuracy has been degraded to resemble the case with $\delta t = 0.025$ and 19 nodes, particularly at the final stages of the evolution. Perhaps the only real improvement is seen near time $t = 0$ where the error approaches zero. This is significant because this initial period of time contains important data regarding the bubble inception process.

In summary, the technique appears sound for generated data of a spherically evolving bubble but loses accuracy as the bubble shape diverges from spherical geometry. Provided that the experimental data depicts a spherical evolution the method will produce reasonable pressure results. Whilst increasing the number of nodes on the bubble surface produces a marked improvement in the accuracy, reduction of the timestep appears not to influence it significantly.

An advantage of using the spherical bubble method is that due to its use of the bubble volume V to compute the mean radius R, any minor fluctuations

on the bubble surface will be averaged out in the process. A corollary of this is that only the gross bubble characteristics such as $\dot{R}$, $\ddot{R}$ and P_b can be found. In order to determine nodewise quantities use of the nodal displacement method is required.

5.2 Nodal displacement method

Progressing from the simple spherical bubble method to the more complex nodal displacement method involves a number of verification stages. The first of these would logically be the comparison between the theoretical nodal normal surface velocities $(\partial\phi/\partial n)_{th}$ which are generated using the mathematical model given in Chapter 3, and the computed $(\partial\phi/\partial n)_c$ extracted from the same coordinate test data using (4.40).

The effect of reducing the timestep from the experimentally equivalent $\delta t = 0.025$ to $\delta t = 0.0125$ is considered in this comparison, as is the effect of increasing the number of nodes from 19 to 37. Figures 5.8 and 5.9 display the nodewise and timewise developments in the value of $\partial\phi/\partial n$ for the theoretical and computed cases, each having a timestep value of $\delta t = 0.025$ and $n = 18$.

We define the relative error in computed $\partial\phi/\partial n$ in the familiar form

$$E_{rel} = \frac{\left|\left(\frac{\partial\phi}{\partial n}\right)_{th} - \left(\frac{\partial\phi}{\partial n}\right)_c\right|}{\left(\frac{\partial\phi}{\partial n}\right)_{th}}, \tag{5.5}$$

and use this as a comparative tool in Figure 5.10. Once again we find that the relative error E_{rel} is not a good indicator of the accuracy of the method and the absolute error, which is given by

$$E_{abs} = \left(\frac{\partial\phi}{\partial n}\right)_{th} - \left(\frac{\partial\phi}{\partial n}\right)_c. \tag{5.6}$$

is utilised. Figure 5.11 shows this error curve.

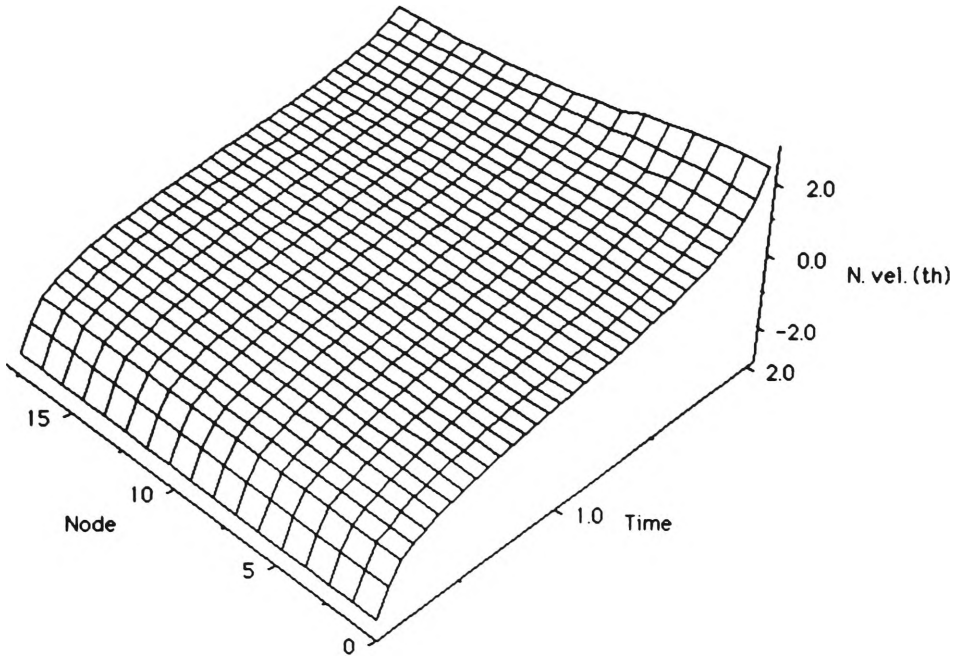

Figure 5.8: Theoretical $\partial\phi/\partial n$ ($\delta t = 0.025$, 19 nodes)

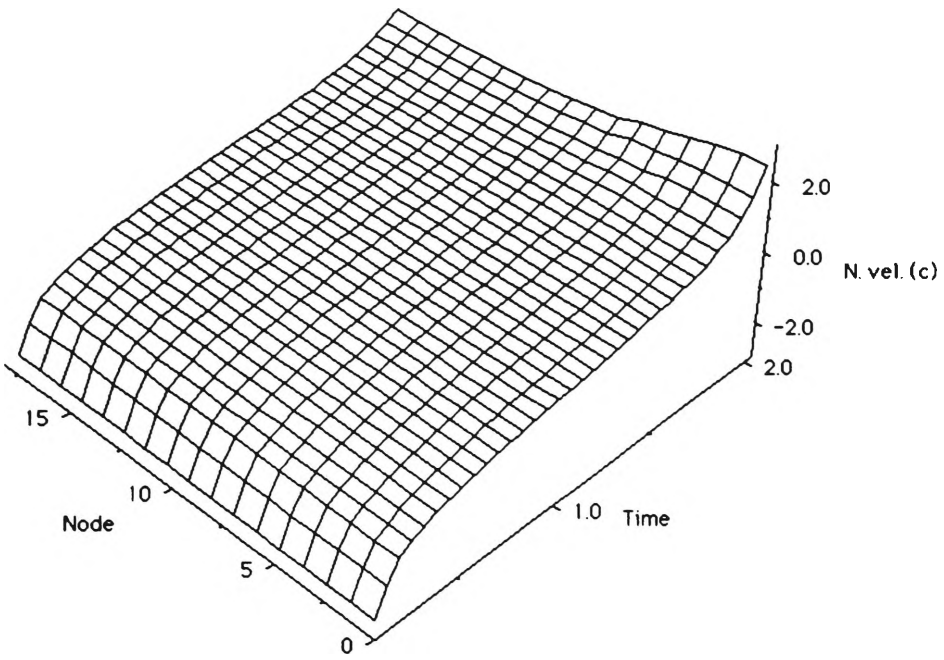

Figure 5.9: Computed $\partial\phi/\partial n$ ($\delta t = 0.025$, 19 nodes)

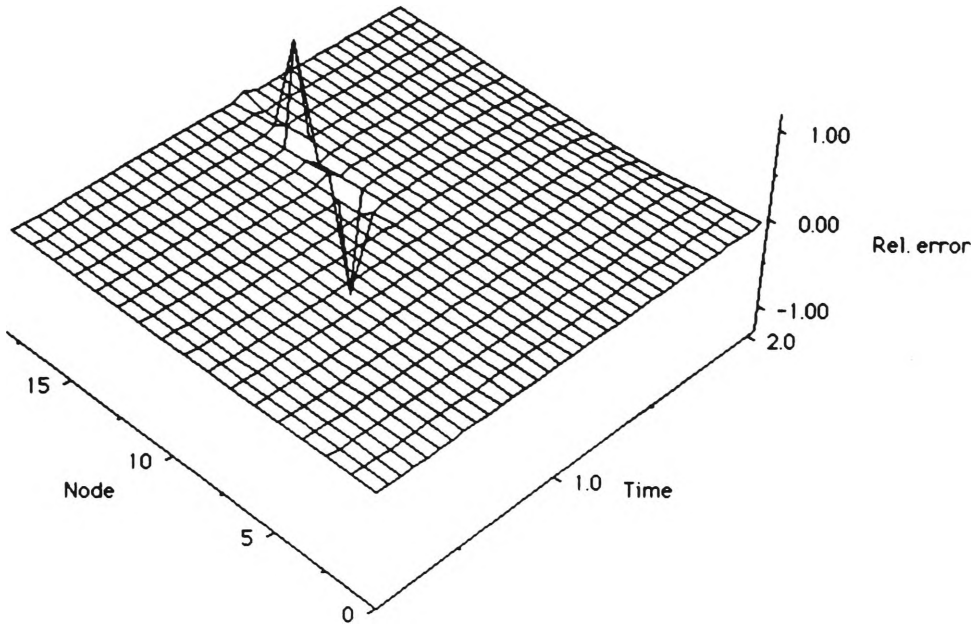

Figure 5.10: Relative error in computed $\partial\phi/\partial n$ ($\delta t = 0.025$, 19 nodes)

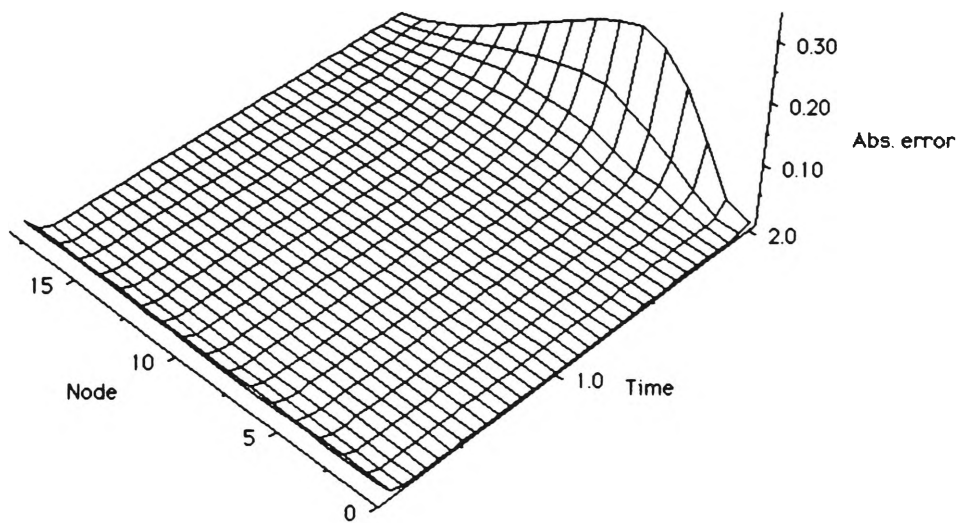

Figure 5.11: Absolute error in computed $\partial\phi/\partial n$ ($\delta t = 0.025$, 19 nodes)

Near $t = 1.0$ the relative error E_{rel} apparently becomes excessive although this is not indicative of poor computation, merely a small denominatior value for $(\partial\phi/\partial n)_{th}$ in (5.5).

A better indicator of the accuracy of the method is found in the plot of E_{abs} in Figure 5.11. Note the reasonably flat surface over the majority of the evolution, verifying that the method is indeed able to accurately compute the normal surface velocities. The largest error is found toward the end of the evolution, where the bubble is collapsing rapidly.

The effect of increasing the number of nodes from 19 to 37 is seen in the absolute error surface given in Figure 5.12.

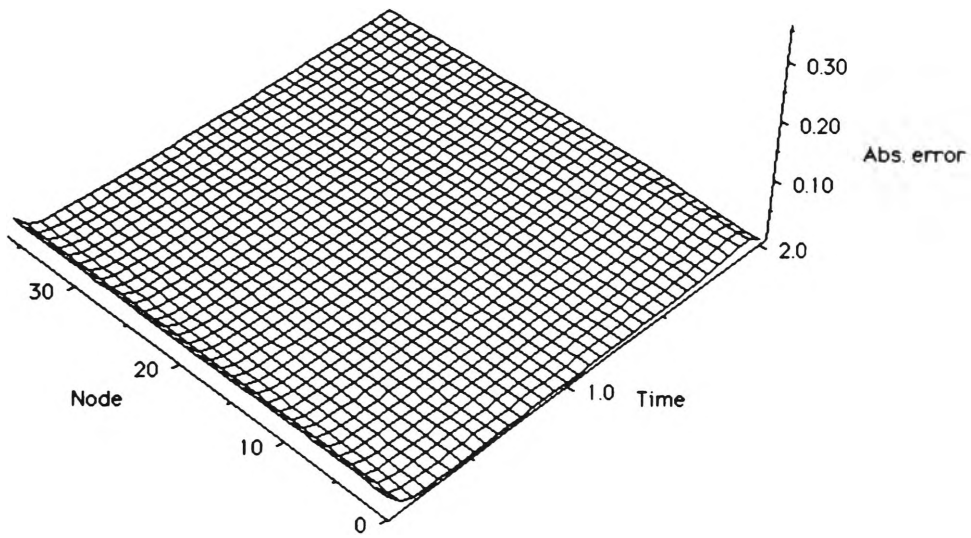

Figure 5.12: Absolute error in computed $\partial\phi/\partial n$ ($\delta t = 0.025$, 37 nodes)

Here we see that the magnitude of the error at the end of the evolution has been substantially reduced, showing that accuracy can be improved by increasing the number of nodes. However at the start of the evolution there is still a noticeable error present. Surface velocities are high in this region and so we would therefore expect that a decrease in timestep δt would reduce the magnitude of the error. Figure 5.12 shows the effect of reducing the timestep by one-half to $\delta t = 0.0125$ while keeping $n = 36$.

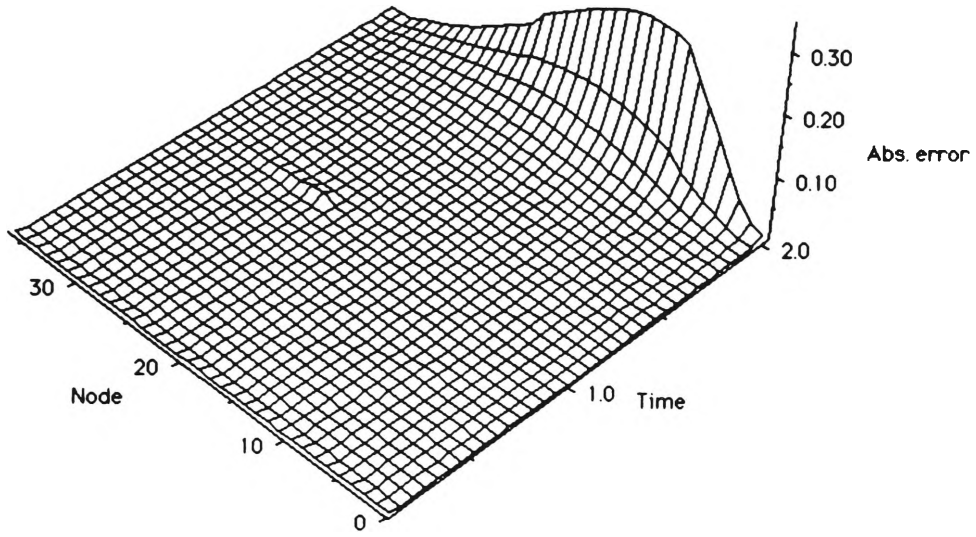

Figure 5.13: Absolute error in computed $\partial\phi/\partial n$ ($\delta t = 0.0125$, 37 nodes)

From Figure 5.13 the method appears to be relatively free from error in the early stages of bubble evolution, indicating that reducing the timestep does achieve greater accuracy. A small area of error is evident around time $t = 1.0$ and this can be attributed to the computational technique failing to determine the nodal normals there. This is because the distance between consecutive bubble surfaces is now one-half smaller than for $\delta t = 0.025$, and as the surface velocity approaches zero there exists some uncertainty as to the position of the consecutive bubble surfaces.

An interesting observation in reducing the timestep is that towards the end of the first collapse E_{abs} becomes excessive, in fact over ten times greater than for $\delta t = 0.025$ with the same number of nodes. This is an unexpected result and can only be attributed to computational error caused by the reduction in timestep.

It becomes clear that a balance must be found between the reduction in timestep and increase in the number of nodes in order to reduce $\partial\phi/\partial n$ errors. Unfortunately in an experimental sense we are restricted to one combination only of δt and n, but it is still a valuable exercise to vary these parameters for verification purposes as was done here.

Because $\partial\phi/\partial n$ and ϕ are inherently linked via the boundary integral method (3.5), the errors in $\partial\phi/\partial n$, and the effects of reduction in timestep

and increase in the number of nodes on these errors will be reflected in the errors in the velocity potential ϕ.

The set of theoretical values ϕ_{th} generated by the verification program can be seen in Figure 5.14, while the computed values ϕ_c are given in Figure 5.15. Note the close correlation between the two surfaces. The comparison between these surfaces again utilises the relative error E_{rel}, shown in Figure 5.16, where

$$E_{rel} = \frac{|\phi_{th} - \phi_c|}{\phi_{th}}. \tag{5.7}$$

We note from Figure 5.16 that the relative error E_{rel} is largest where the value of the demoninator term in (5.7) is small, that is, near $t = 1.0$. However the surface is otherwise flat with a relative error close to zero, clearly displaying the accuracy of the nodal displacement method using the experimentally equivalent values of $\delta t = 0.025$ and $n = 18$.

A better indication of the accuracy can be found in the absolute error E_{abs},

$$E_{abs} = \phi_{th} - \phi_c, \tag{5.8}$$

which appears to be constant in the nodewise direction in Figure 5.17 but varying with time. The largest errors can be found toward the end of the first collapse of the theoretical bubble, after time $t = 1.75$. The method appears to be most accurate near time $t = 1.0$ where E_{abs} approaches zero. This is significant because it is at this point in time when the greatest uncertainty in the position of the bubble surfaces exists.

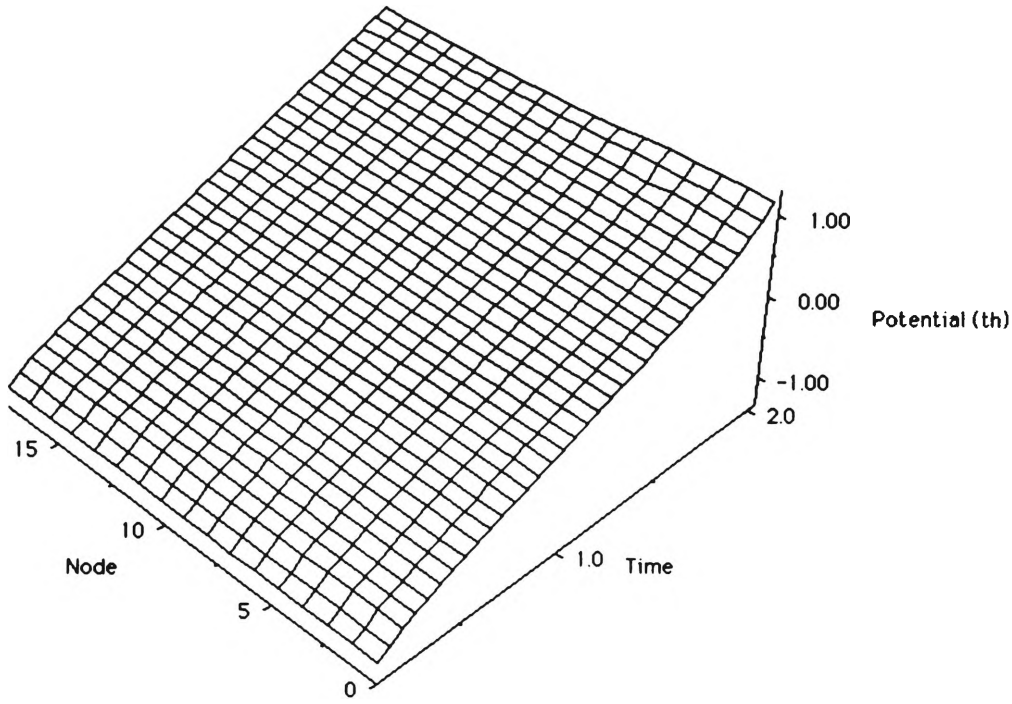

Figure 5.14: Theoretical ϕ ($\delta t = 0.025$, 19 nodes)

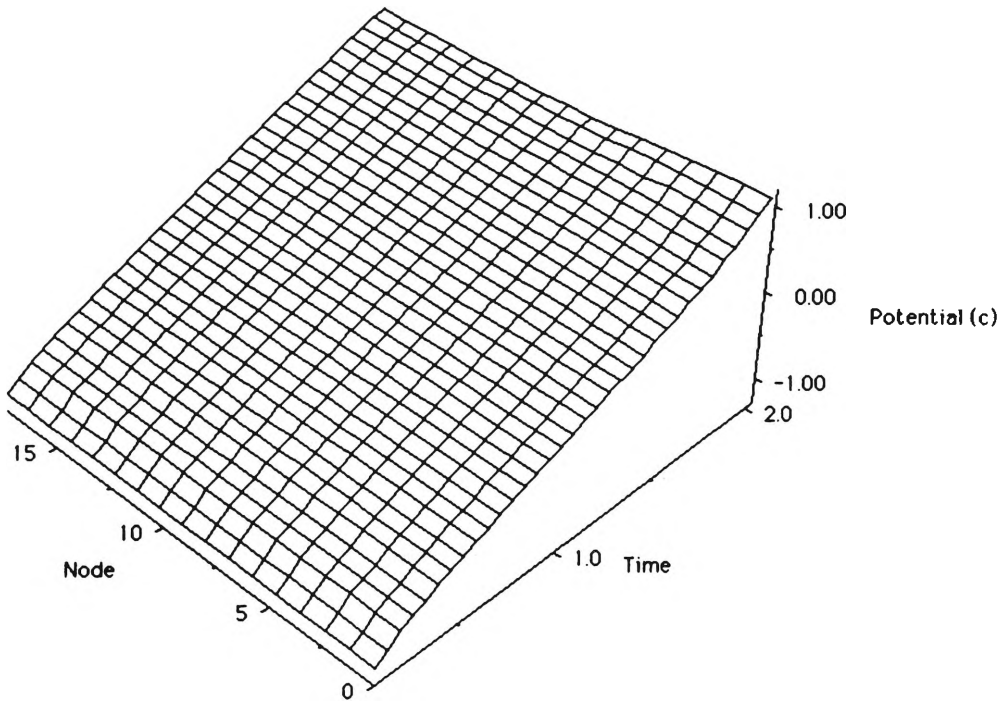

Figure 5.15: Computed ϕ ($\delta t = 0.025$, 19 nodes)

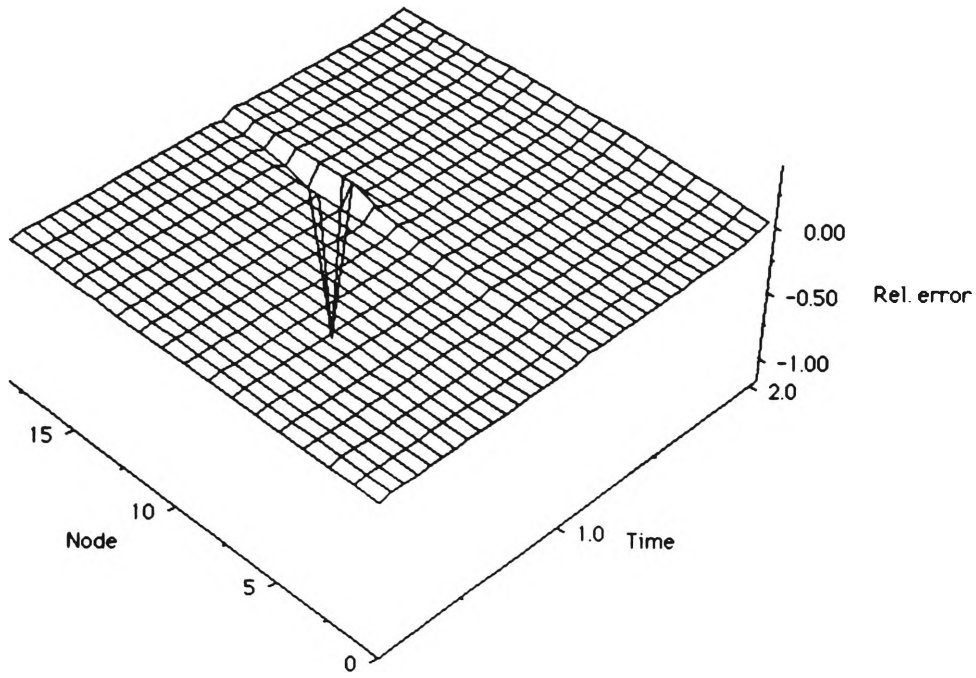

Figure 5.16: Relative error in computed ϕ ($\delta t = 0.025$, 19 nodes)

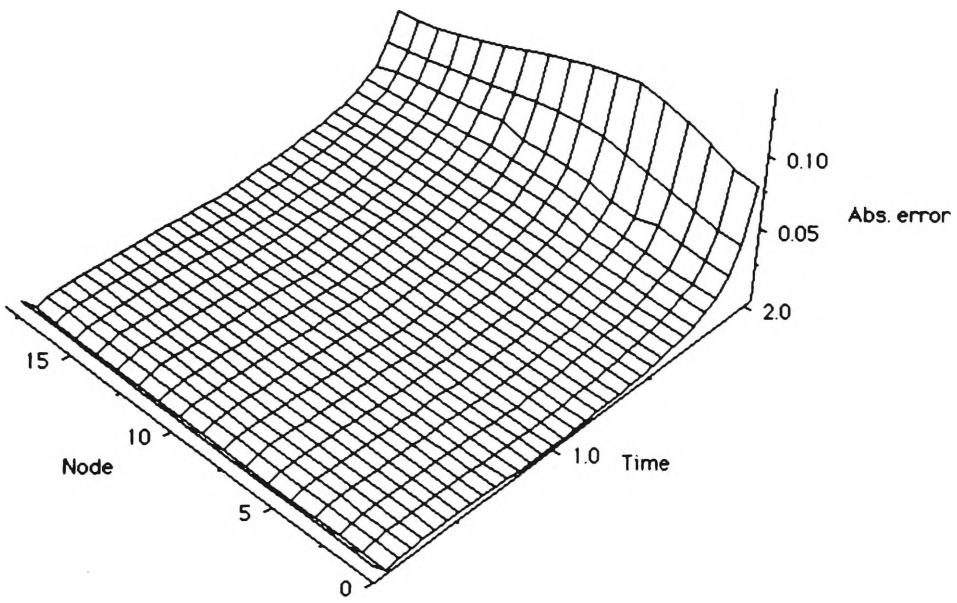

Figure 5.17: Absolute error in computed ϕ ($\delta t = 0.025$, 19 nodes)

Consider now the effect that increasing the number of nodes to 37 and subsequently reducing the timestep to $\delta t = 0.0125$ has on the relative error. Figures 5.18 and 5.19 show the result of these changes.

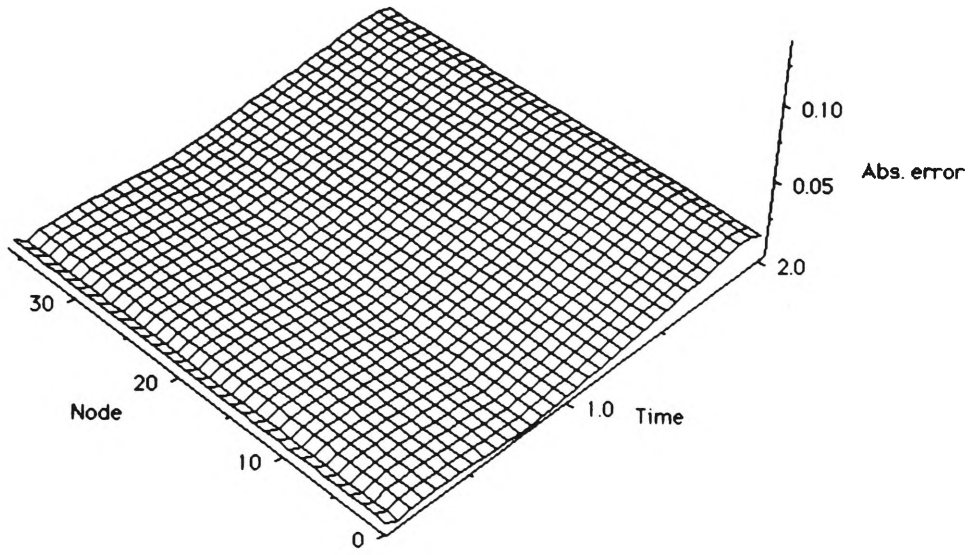

Figure 5.18: Absolute error in computed ϕ ($\delta t = 0.025$, 37 nodes)

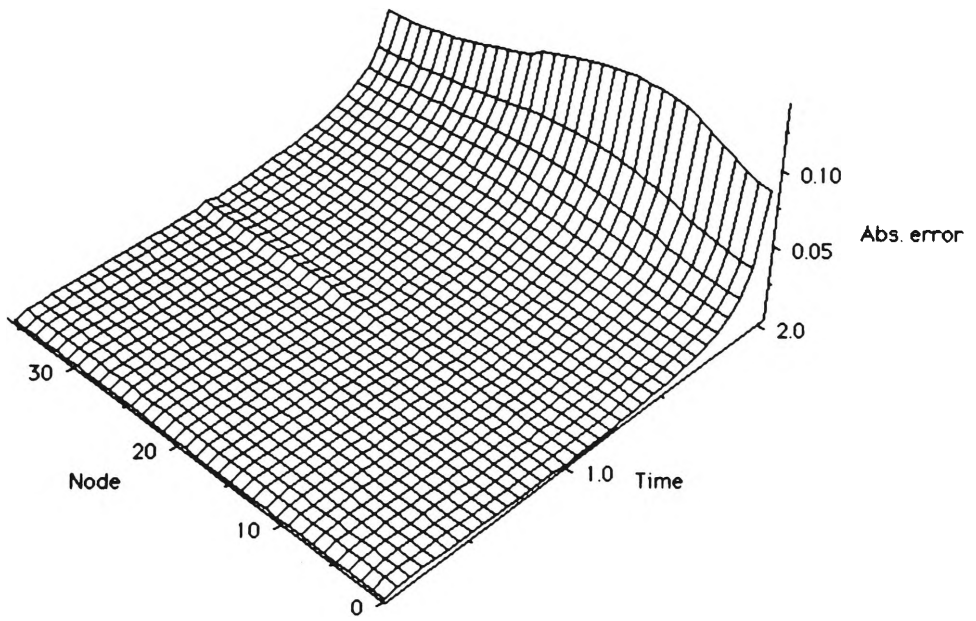

Figure 5.19: Absolute error in computed ϕ ($\delta t = 0.0125$, 37 nodes)

The trends in the absolute error in $\partial\phi/\partial n$ are reflected in that of ϕ whereby an increase in the number of nodes is accompanied by an increase in accuracy, whilst a subsequent decrease in timestep clearly increases the errors in the final stages of the first collapse. Once again it can be said that a trade-off is necessary between timestep and number of nodes to yield the best possible results.

Upon application of the modified Bernoulli equation (4.65) to the potential ϕ and its derivatives, the computed pressure is found on a nodewise basis. The governing equation (5.2) for the theoretical model predicts that the pressure p_{th} on the surface of the bubble should be equal at each node. Figure 5.20 shows this trend over time. However we find that for the same combination of timestep/number of nodes the values for the computed pressure p_c in Figure 5.21 are produced.

The main area of discrepancy between the theoretical and computed pressures is around the point of minimum surface velocity at $t = 1.0$. Here it is clear that the nodal displacement method computes a defined pressure peak for each node, whereas theoretically this should not exist. We can surmise that the error in ϕ is the dominant reason for this discrepancy, since the error in computed $\partial\phi/\partial n$ is much less pronounced here.

More significantly, the theoretical normal surface velocity $(\partial\phi/\partial n)_{th}$ is decreasing nodewise at the end of the first collapse (Figure 5.8) and so we would expect to see a corresponding increase in nodewise pressure at this time. The nodal displacement method results show this increase while the theoretical results do not, revealing one weakness of assuming a theoretically constant nodewise pressure distribution throughout the complete bubble evolution.

This weakness is best illustrated by examining the absolute error E_{abs} in computed pressure using

$$E_{abs} = p_{th} - p_c, \tag{5.9}$$

which is depicted graphically in Figure 5.22.

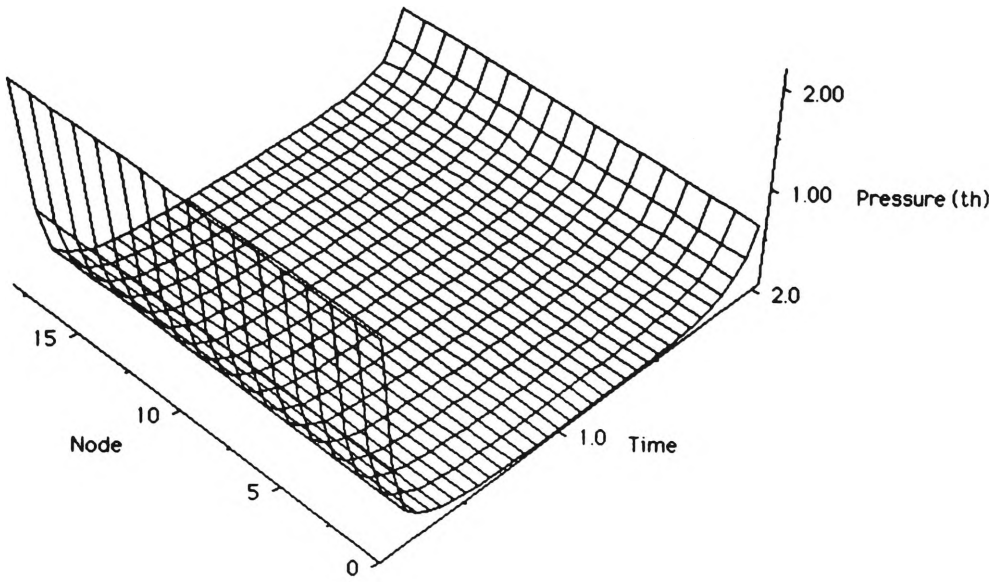

Figure 5.20: Theoretical pressure p_{th} ($\delta t = 0.025$, 19 nodes)

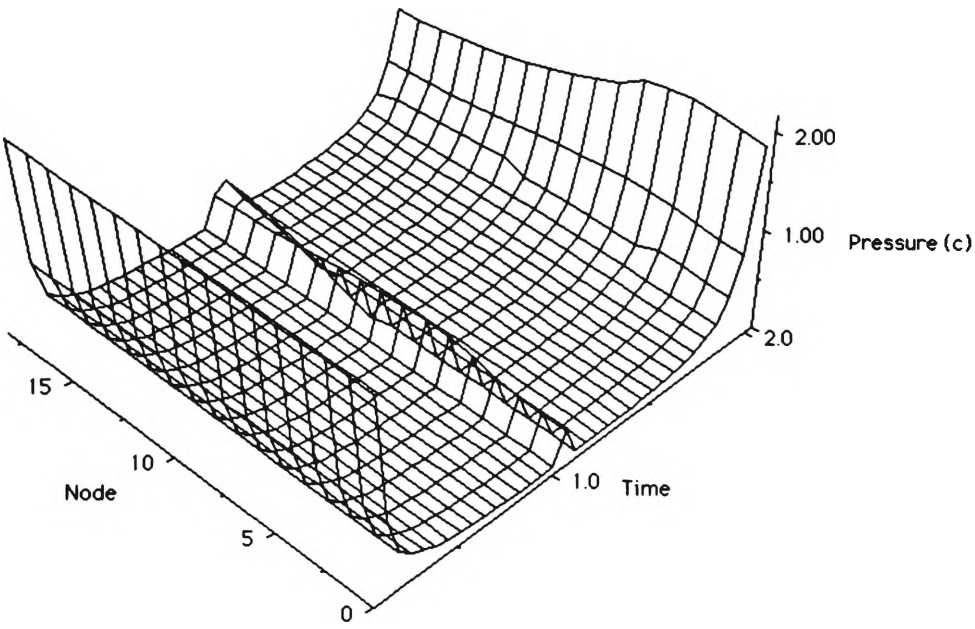

Figure 5.21: Computed pressure p_c ($\delta t = 0.025$, 19 nodes)

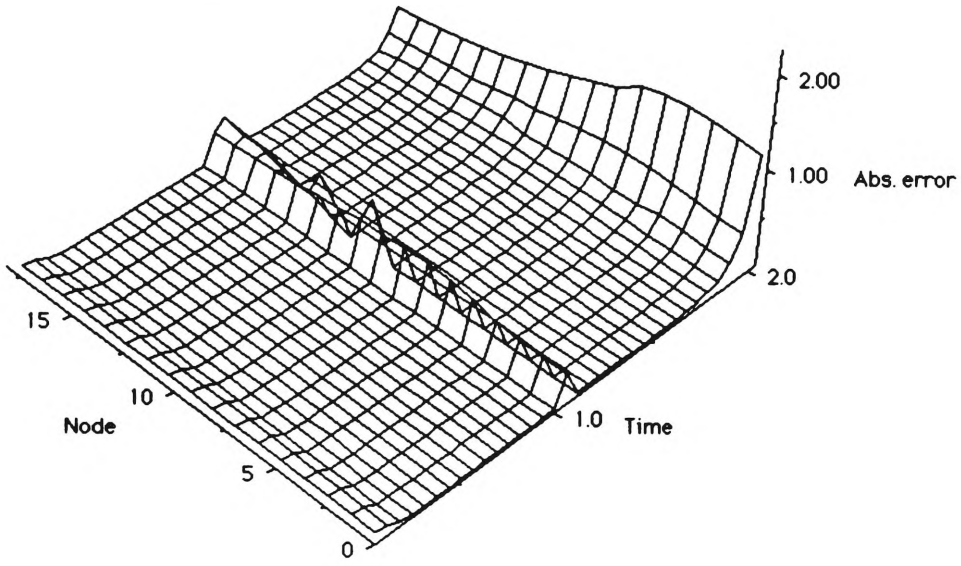

Figure 5.22: Abs. error in computed pressure p_c ($\delta t = 0.025$, 19 nodes)

Consider also the relative error E_{rel} in the computed pressure which is given by

$$E_{rel} = \frac{|p_{th} - p_c|}{p_{th}}, \tag{5.10}$$

and is shown in Figure 5.23. Here it can be seen that the main difference between the theoretical and computed pressures occurs around time $t = 1.0$. Due to the relative smallness of the denominator term in (5.10) this difference is exaggerated, yielding a result which is unrepresentative of the true error there.

If the number of nodes is now increased to 37, we see that the absolute error is reduced overall, particularly toward the end of the first collapse. Figure 5.24 shows this improvement in accuracy.

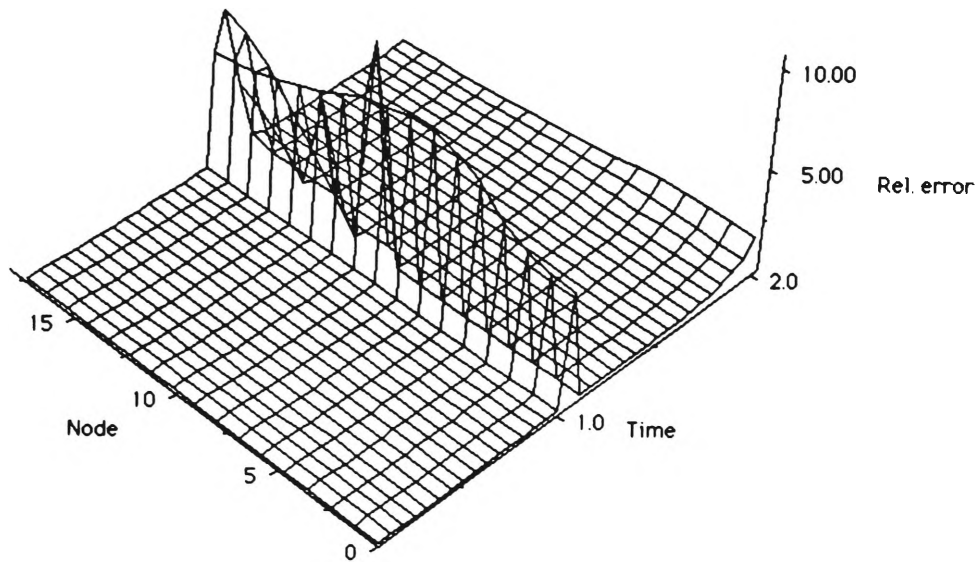

Figure 5.23: Rel. error in computed pressure p_c ($\delta t = 0.025$, 19 nodes)

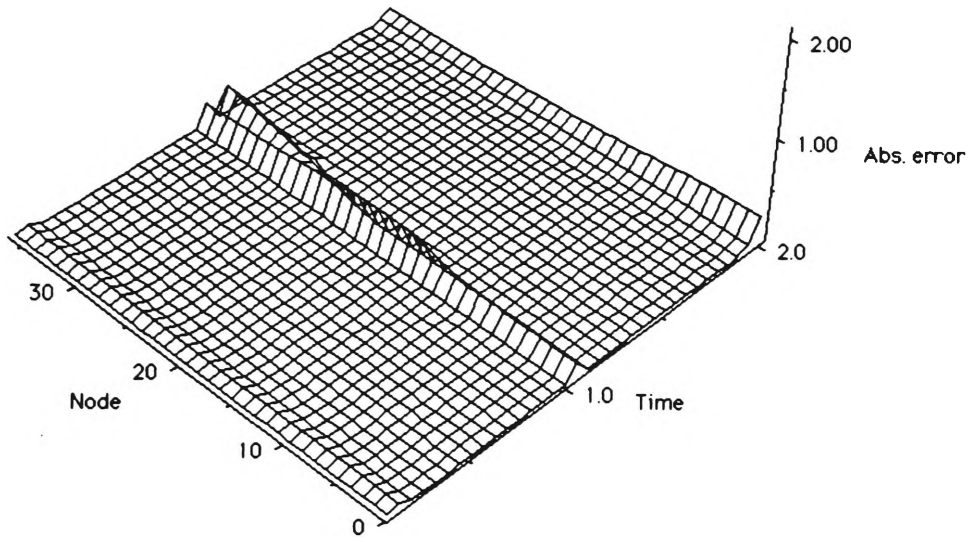

Figure 5.24: Abs. error in computed pressure p_c ($\delta t = 0.025$, 37 nodes)

Upon reduction of the timestep from $\delta t = 0.025$ to $\delta t = 0.0125$, shown in Figure 5.25, it becomes obvious that the absolute error has increased again.

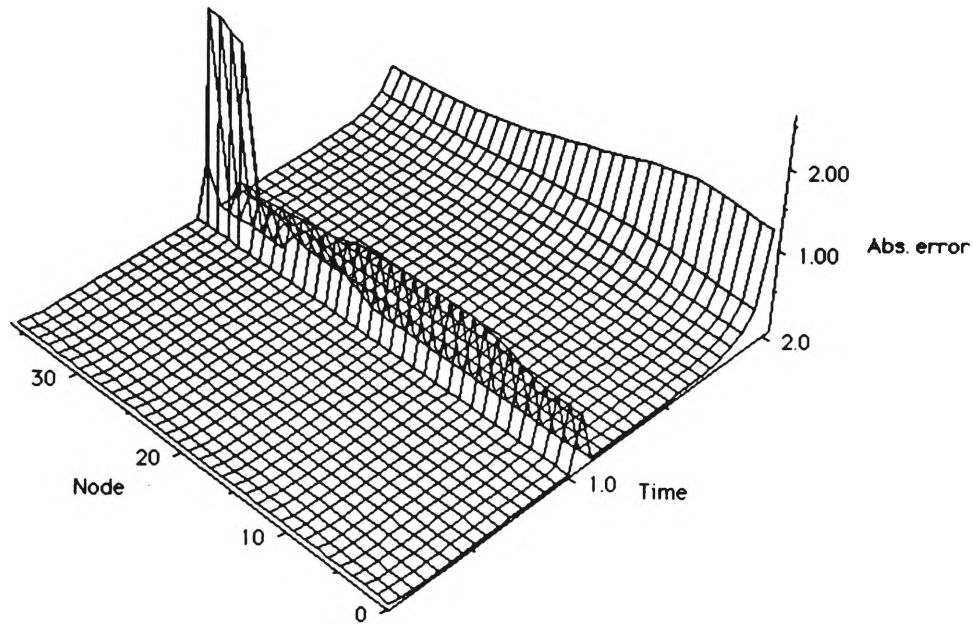

Figure 5.25: Abs. error in computed pressure p_c ($\delta t = 0.0125$, 37 nodes)

As with the other computed quantities used to derive the computed pressures, reduction in the size of the timestep could be viewed as having a detrimental effect on the accuracy of the method.

One indication confirming the this is the large error peak occurring near the 37th node at about time $t = 1.0$. Since this peak appears unrepresentative, in comparison with the errors shown in Figure 5.24, it could be argued that reducing the timestep does reduce the method's accuracy. However, due to the uniform nodewise pressure distribution assumed in the theoretical data, it is unclear if this is in fact the case.

It would be appropriate at this point to summarise the accuracy of the nodal displacement method by means of a comparative graph. Figure 5.26 shows the effects of varying both the timestep and number of nodes on the maximum nodal absolute error in computed pressure.

The most accurate combination appears to be with $\delta t = 0.025$ and 37 nodes, where the absolute error is the lowest of the three combinations when averaged over time. Fortunately $\delta t = 0.025$ is equivalent to the experimental timestep used, so with an increase in the number of experimental nodes from the current 19 to 37, better accuracy can be expected.

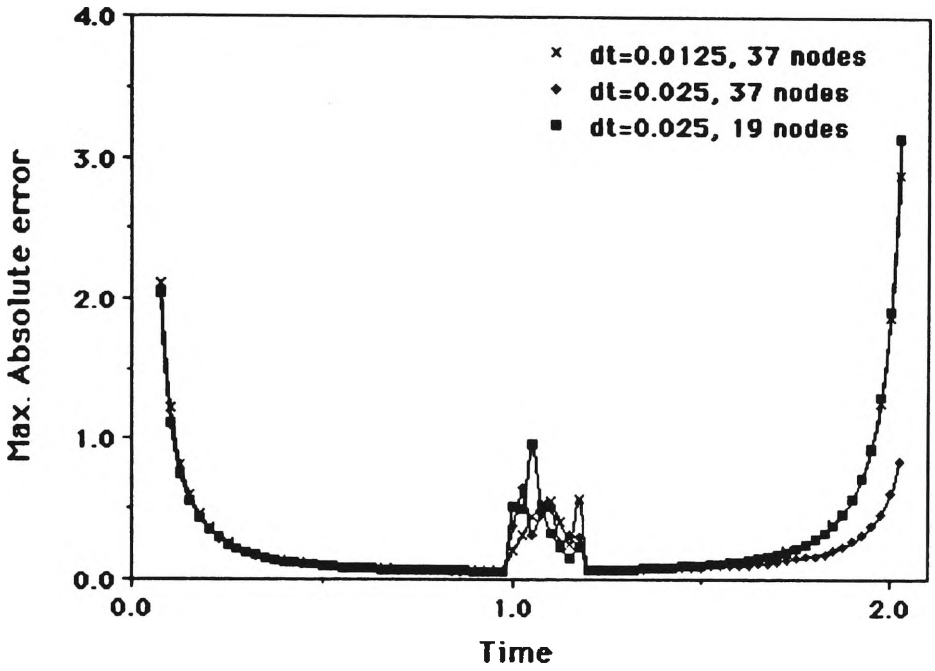

Figure 5.26: Max. nodal abs. errors in computed pressure p_c

In summary, then, the validation of the pressure computing techniques has been carried out using generated data with varying timesteps and numbers of nodes. Both are succeptible to errors and their accuracies are strongly influenced by data quality.

It was found that the nodal displacement method was better suited to data with a minimum of experimental noise in the coordinate sets. This is because at the end of the first expansion the normal surface velocity approaches zero, and its sign fluctuates depending on the nodal positions in space. This produces some errors in $\partial\phi/\partial n$ and these are more prevalent with unsmooth or noisy data.

A reduction in the size of the timestep does not necessarily increase accuracy since the consecutive bubble surfaces will be closer together, again making determination of $\partial\phi/\partial n$ less accurate. Conversely, an increase in the number of nodes was seen to yield a noticeable improvement in accuracy.

One advantage of this method is its ability to analyse data from a distorted bubble such as one in the final stages of collapse (Figure 5.1). Clearly this is a more realistic situation than one that assumes purely spherical motion.

The spherical bubble method, on the other hand, is not as succeptible to minor surface pertubations as the nodal displacement method since it examines only the quantities of gross bubble motion. Increasing the number of nodes was seen to be accompanied by an increase in its accuracy, whilst reducing the timestep tended to have an adverse effect toward the end of the first collapse. Although the accuracy of the method decreases rapidly in the final stages of collapse when the bubble diverges from a spherical geometry, it is still a useful method and is well suited for use in experimental analysis.

Chapter 6

RESULTS AND DISCUSSION

Having validated the computational techniques used in this work, we will now proceed to apply them to real experimental data. In this chapter the shape histories of a spark-induced vapour bubble in the vicinity of a rigid boundary from three separate experiments are analysed and the results presented in graphical format.

The first experimental photographic record, film RB3, was an early work and the original film contains a large amount of optical noise. In addition, the image analysis procedure introduced further noise to the raw coordinate files resulting in poor data for use in the computational techniques. Nethertheless some interesting results are obtained.

The abscence of any appreciable experimental noise in the newer second RBS30 and third RBS31 films shows clearly the recent improvements made in both the experimental procedures and the digital image analysis, while utilising the same apparatus and framing rate as the earlier film RB3.

Comparisons between the three experiments, based on these results, is made, as well as comparisons with the theoretical and experimental work of others. It will be seen that the spherical bubble and nodal displacement methods are well suited to experimental analysis, both enabling important quantities to be unobtrusively extracted from the original photographic data.

6.1 Film RB3

Using the apparatus and parameters described in Section 2.2, the film RB3 was made. A summary of the experimental parameters and errors is given in Table 6.1.

Parameter	Symbol	Value
film title		RB3
date of filming		June 1991
framing rate		6000 /sec
time betweeen frames		166.5μs
analysed timestep	δt	333μs
discharge voltage	V_d	10000 V
discharge capacitance	C_d	1 μF
ambient free surface pressure	p_a	5000 Pa
error in ambient free surface pressure		250 Pa
depth of inception	H	200.0 mm
error in depth of inception		1.0 mm
hydrostatic pressure at inception point	p_∞	6962.0 Pa
error in pressure at inception point		250 Pa
vapour pressure	p_v	1704.0 Pa
error in vapour pressure		55.8 Pa
distance from rigid boundary	h_r	50.0 mm
error in distance from rigid boundary		0.5 mm
bubble orientation		above boundary
water temperature	T_w	15.0°C
error in water temperature		0.5°C
maximum radius	R_m	24.5 mm
error in maximum radius		0.8 mm
standoff parameter	γ	2.041
buoyancy parameter	δ	0.270
frames/first pulsation		136
period of first pulsation		0.0228 sec
scale		0.19157 mm/pixel
error in radius measurement		3 pixels

Table 6.1: Experimental parameters and errors for film RB3

An example sequence of raw images taken from this film is shown in Figure 6.1.

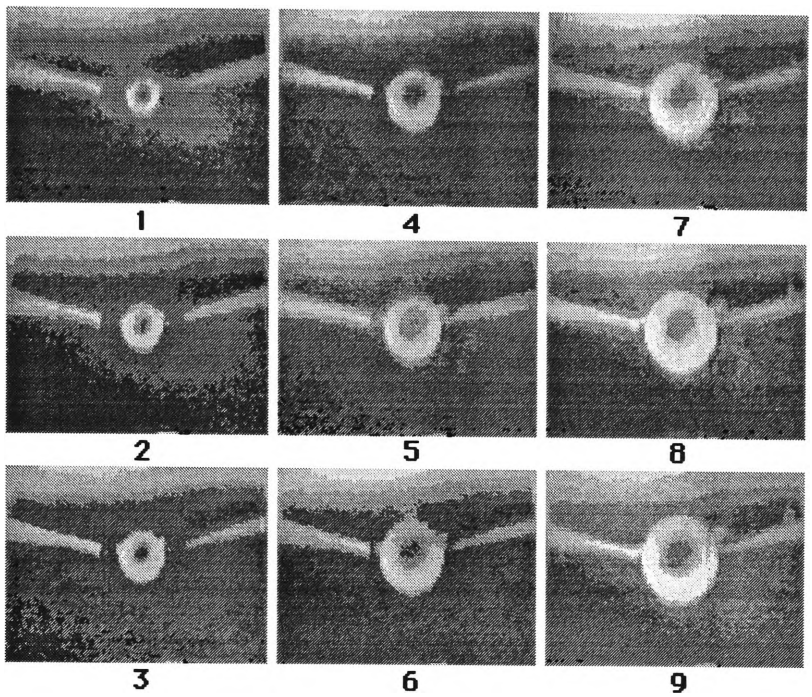

Figure 6.1: Sequence of raw images from film RB3

Digital image analysis of this film and conversion into the form shown in Figure 4.2 was undertaken by Mr Darren Weise at the Defence Science Technology Organisation using the technique described in Appendix A. Every alternate film frame was analysed, resulting in 165 Cartesian coordinate files recording the complete evolution of the bubble, each frame being $333\mu s$ apart in time. Using the graphic display program in Appendix D, this evolution was plotted in Figure 6.2. For clarity, only every fifth recorded file is plotted. The rigid boundary, represented by a straight line, is located at the bottom of the figure.

A preliminary observation is that there is a small horizontal displacement of the bubble centroid toward the left in the final stages of the evolution. This has been neglected by assuming that all dominant forces are acting in the vertical direction, thus allowing the data preprocessing program in Appendix B to yield the axisymmetric data in the cylindrical coordinate form of Figure 4.4.

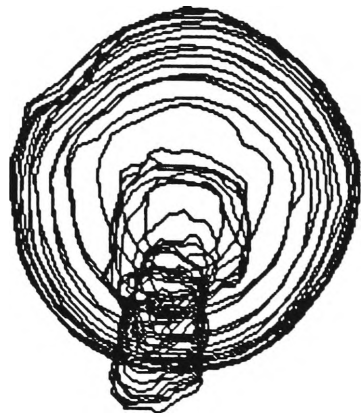

Figure 6.2: Shape history, complete evolution, film RB3

The occurrence of the numerous pulsations of the bubble during its evolution is also evident from the plot of volume against time in Figure 6.3. The bubble volume is computed using (4.94).

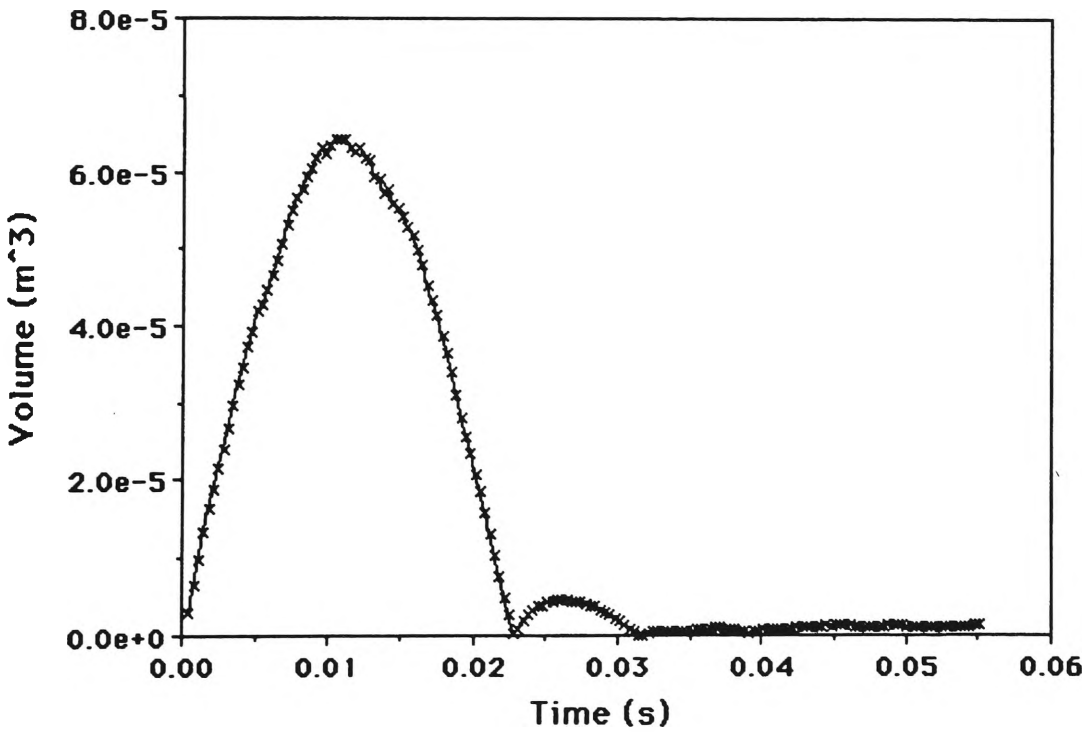

Figure 6.3: Volume V vs time, complete evolution, film RB3

In Figure 6.3 we see that the first pulsation of the evolution is marked by a much larger maximum bubble volume (0.0000643 m^3) than in subsequent cycles. Since volume varies as R^3, where R is the mean bubble radius at a given time, a small reduction in R would correspond to a much larger reduction in volume. This can be seen in the plot of R vs time given in Figure 6.4. The mean radius R is calculated from the bubble volume V using (4.14). The presence of the numerous pulsations of the bubble are also evident from this graph.

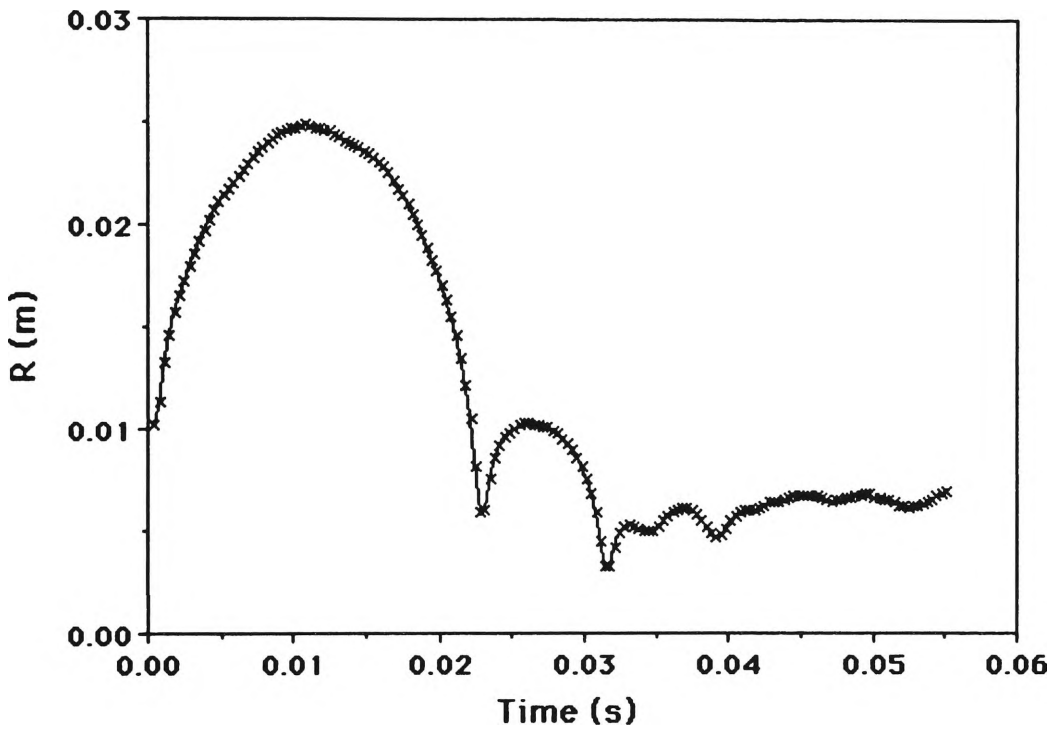

Figure 6.4: Radius R vs time, complete evolution, film RB3

The maximum radius $R_m = 24.5$ mm of this experiment compares favourably with that obtained by Gibson (1968) using the same apparatus, where R_m was measured as 15.9 mm. The smaller maximum radius can be attributed to the smaller quantity of electric spark energy (8 J) used in his experiment (Gibson, 1972). The available spark energy E_{spark} for the present film RB3 can be calculated from Gibson's (1972) formula

$$E_{spark} = \frac{1}{2} C_d V_d^2, \tag{6.1}$$

where from Table 6.1 C_d, the discharge capacitance, is 1μF and V_d, the

discharge voltage, is 10000 V. This results in $E_{spark} = 50$ J, a much higher figure.

The fact that bubbles of this size can be generated reveals one advantage of spark-discharge methods over laser methods. For example, a maximum bubble radius of only 4.5 mm was produced by Vogel et. al. (1989) using a Q-switched ruby laser, which could only avail 400 mJ of energy for the bubble formation process.

A detailed plot of the more relevant first pulsation is given in Figure 6.5. Here, every recorded raw coordinate file is plotted.

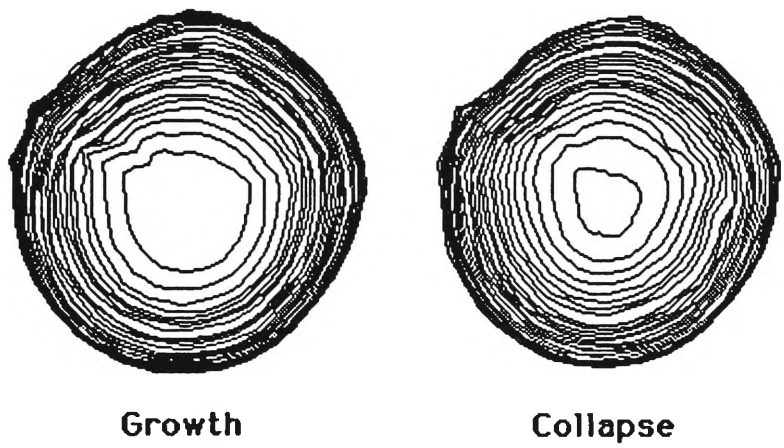

Figure 6.5: Shape history, first pulsation, film RB3

Note the distinct contrast between the above figure and the much smoother surface of the theoretical bubble evolution shown in Figure 5.1. The deviation from a spherical shape indicates that there is an appreciable amount of experimental and image analysis noise present in the raw Cartesian coordinate files.

An interesting observation is the apparent absence of centroidal movement toward the rigid boundary in the first pulsation. A good indication of this may be in the value of the product $\gamma\delta = 0.551 > 0.442$ which would suggest a movement away from the boundary if at all. This correlates well with the above figure.

The files were converted to cylindrical coordinates using the data preprocessing program (Appendix B) and subsequently analysed using the pressure extraction program (Appendix C) The nondimensional scaling factors used in this process, as described in Chapter 4, are given in Table 6.2.

Parameter	Symbol	Value
no. raw data nodes	$npts$	36
no. processed nodes-1	n	18
length scale	R_m	0.0263
time scale	$R_m(\rho/\Delta p)^{\frac{1}{2}}$	0.01174
velocity scale	$(\Delta p/\rho)^{\frac{1}{2}}$	2.236
pressure scale	Δp	5257.0
volume scale	R_m^3	0.000018
energy scale	$\frac{1}{2}R_m^3\Delta p$	0.0452

Table 6.2: Nondimensional scaling factors for film RB3

The kinetic energy of the evolving bubble can be determined by using (4.98). Figure 6.6 displays the variation of this quantity over time.

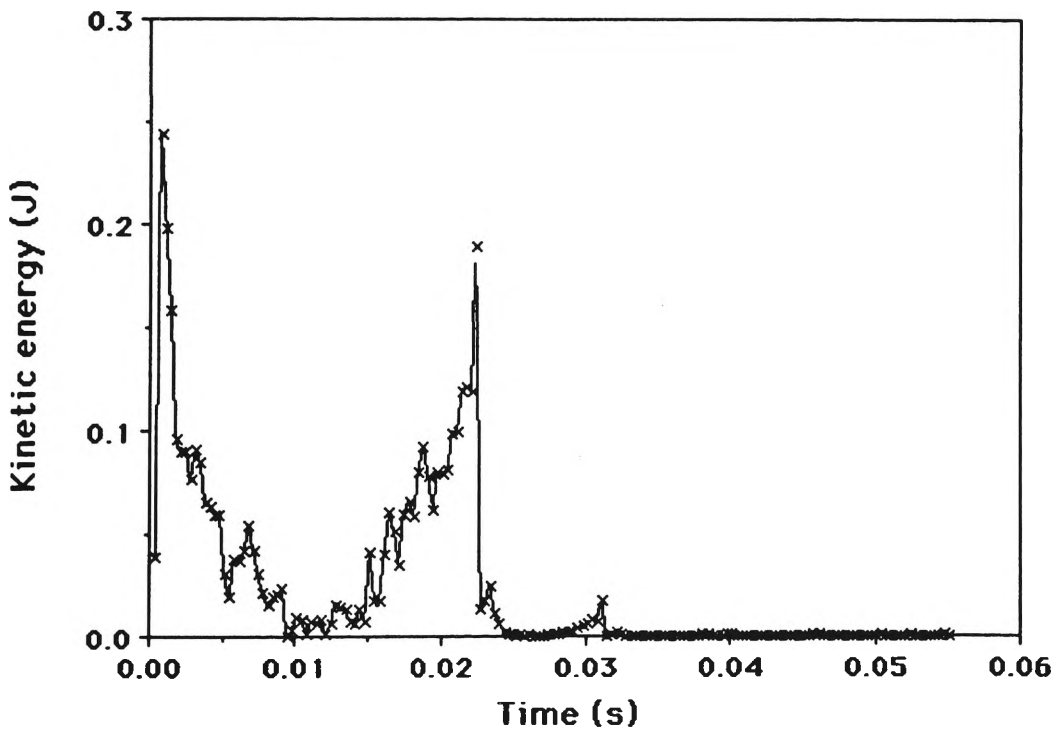

Figure 6.6: Kinetic energy E_k vs time, complete evolution, film RB3

From this plot, it is clear that energy is lost to the surrounding fluid during the course of the evolution. This can be seen in the reduction in the energy peaks at the end of each collapse. The presence of experimental noise is evident from the spurious nature of the curve in between the energy peaks.

Gibson (1972) states that only a maximum of 10 percent of the electrical energy is dissipated in the spark, and for the present work this equates to 5 J. From Figure 6.6 we see that the initial kinetic energy of the bubble is 0.242 J. If we assume that 5 J is liberated by the spark, then the balance of the energy is most likely lost in the form of the latent heat of formation of the water vapour, and other heat losses.

We now examine the behaviour of the quantities used by the spherical bubble method in (4.13) over the first pulsation to compute the bubble pressure p_b, namely the mean radius R, average surface radial velocity $\dot{R}$, and mean surface radial acceleration $\ddot{R}$. These quantities are computed by the pressure extraction program from the preprocessed data. The quantities R and $\ddot{R}$ are smoothed timewise to reduce the effects of noise, the smoothing scheme being based on the three-point algorithms

$$R_{t,smoothed} \Rightarrow \frac{1}{4}(R_{t-\delta t} + 2R_t + R_{t+\delta t}), \tag{6.2}$$

and

$$\ddot{R}_{t,smoothed} \Rightarrow \frac{1}{4}(\ddot{R}_{t-\delta t} + 2\ddot{R}_t + \ddot{R}_{t+\delta t}), \tag{6.3}$$

where the subscript t denotes the time and δt is the timestep. Figure 6.7 depicts the behaviour of R for the first pulsation, and Figure 6.8 $\dot{R}$ for the same time span.

Even after smoothing, the presence of experimental noise can be seen in the somewhat jagged appearance of the curves, although the $\dot{R}$ curve does possess a defined linear region. As expected, the maximum surface velocity, at 6.863 m/sec, occurs at the start of the first expansion phase.

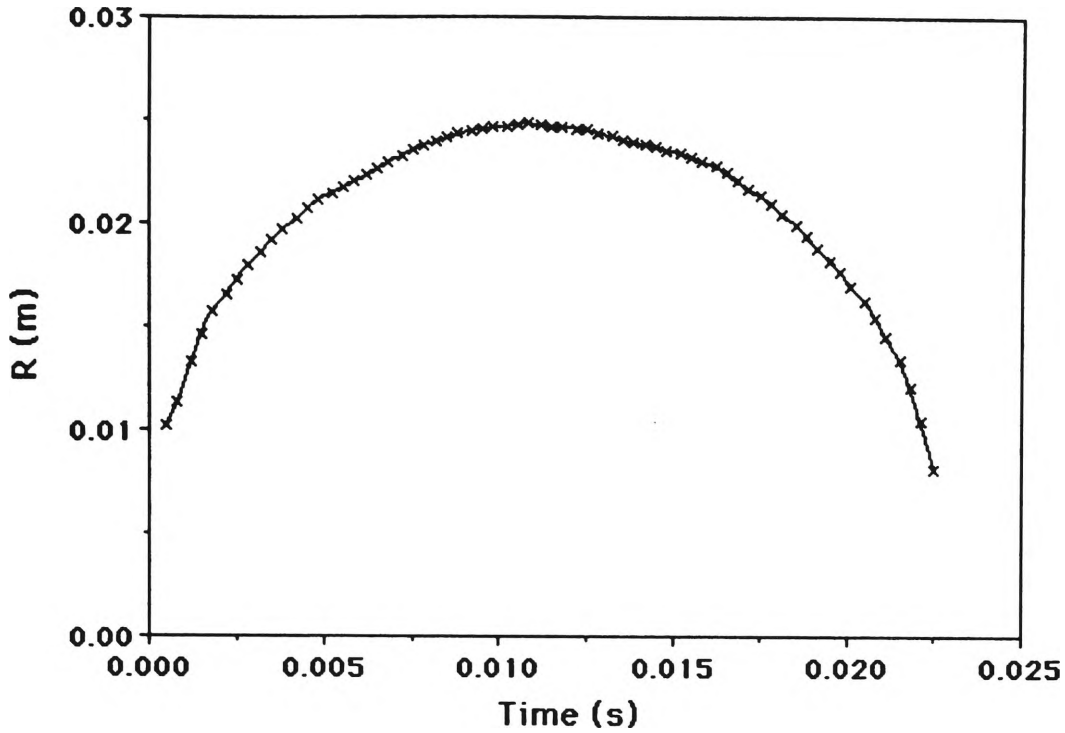

Figure 6.7: Radius R vs time, first pulsation, film RB3

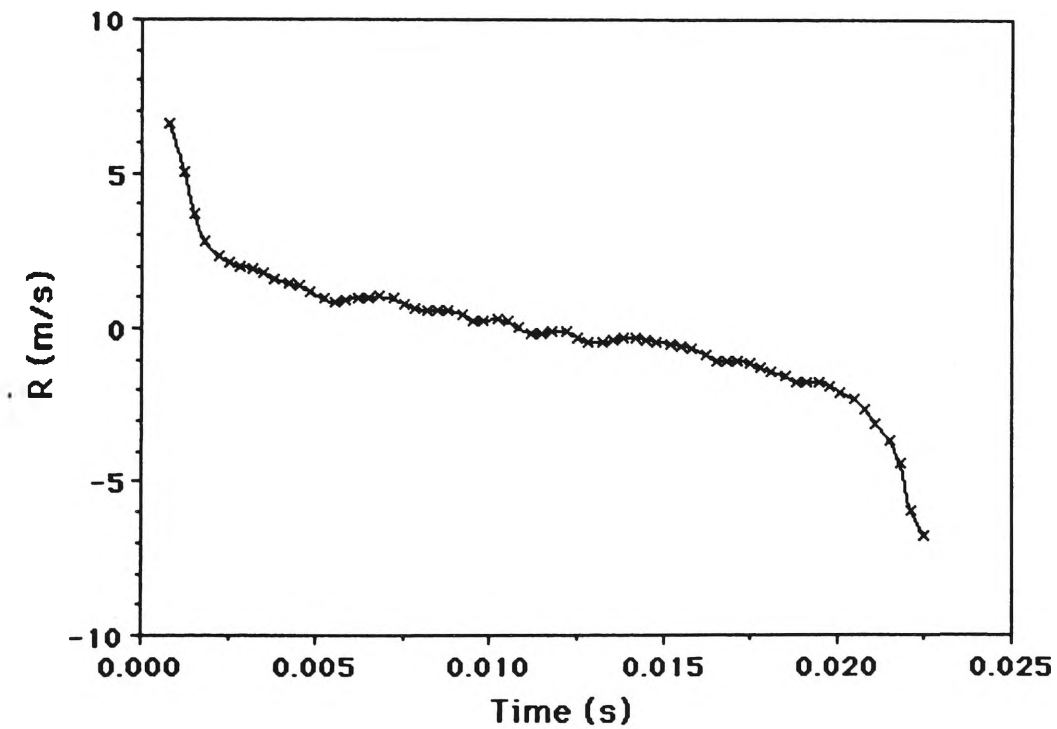

Figure 6.8: Radial velocity $\dot{R}$ vs time, first pulsation, film RB3

At maximum volume, at time $t = 0.0104$ sec, the surface velocity changes sign and becomes negative, indicating the start of the first collapse. At the end of the this collapse the linear trend in velocity disappears as the bubble accelerates toward the point of minimum volume.

The loss in kinetic energy is evident from the smaller magnitude of the surface velocity, which at the final time $t = 0.0221$ sec is 5.335 m/sec. This value compares favourably with the results of Vogel et. al. (1989) who quoted velocities in the order of 7 m/sec in their examination of much smaller laser-produced cavitation bubbles. Considering the physical differences between the size and generation techniques of these two types of bubbles, this is an interesting result.

The mean surface radial acceleration $\ddot{R}$ is shown for the first pulsation in Figure 6.9.

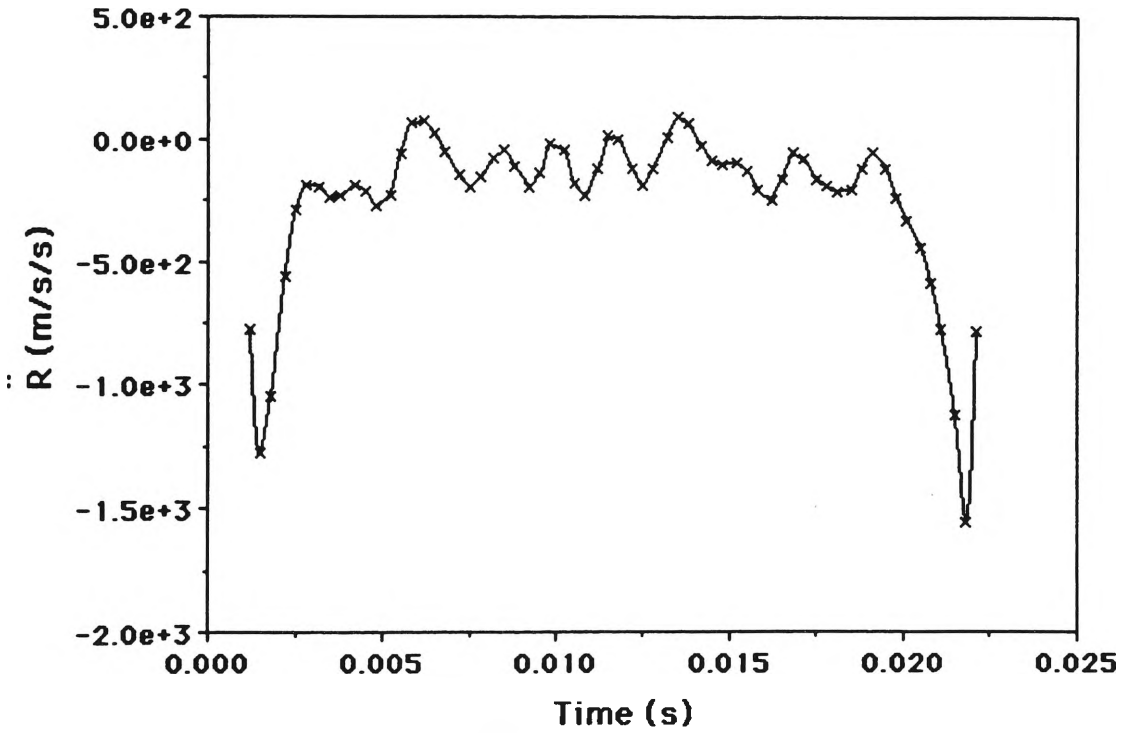

Figure 6.9: Radial acceleration $\ddot{R}$ vs time, first pulsation, film RB3

Clearly the effects of experimental noise still persist. By virtue of the method used in its determination, namely finite difference, any fluctuation in the smoothness of $\dot{R}$ will be magnified in $\ddot{R}$.

This magnification of instabilities is unfortunate since they are further reflected in the behaviour of the bubble pressure p_b calculated using (4.13), which is repeated here.

$$p_b = R\ddot{R} + \frac{3}{2}\dot{R}^2 + \mu R[R\ddot{R} + 2\dot{R}^2] + 1. \tag{6.4}$$

The resulting values of p_b are smoothed using the same algorithm as is used for R and $\ddot{R}$ in (6.2) and (6.3). These are plotted in Figure 6.10 along with the theoretical pressure p_{th} generated using (5.2). Note the dominance of $\ddot{R}$.

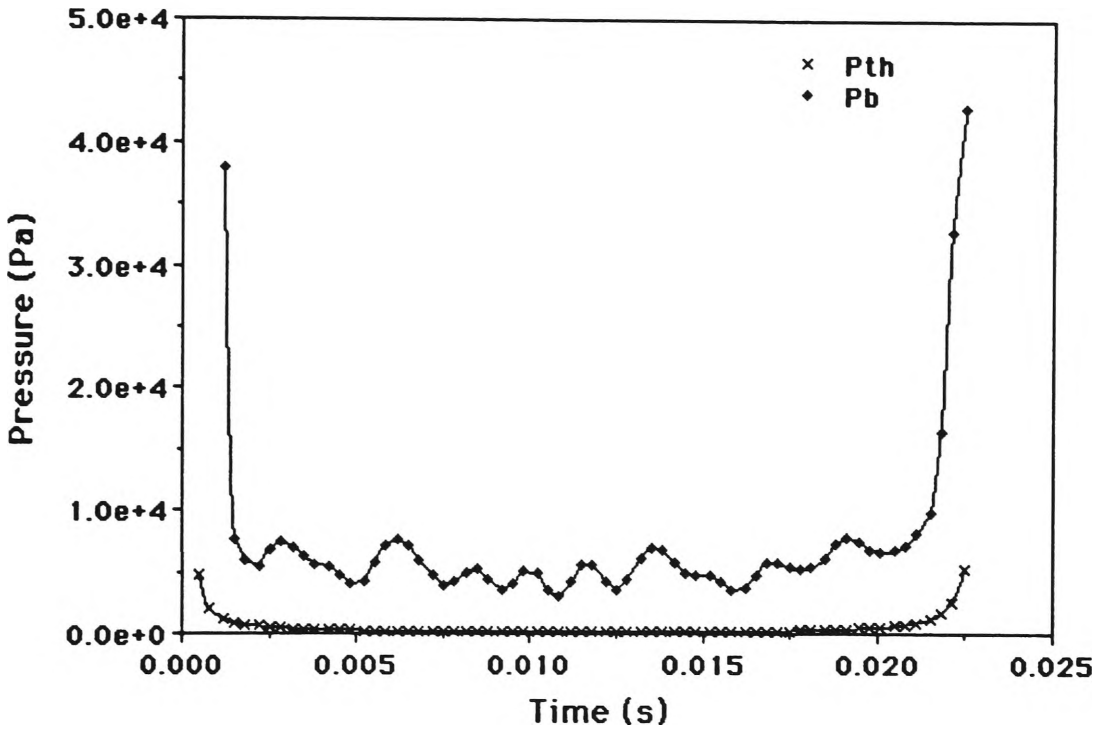

Figure 6.10: Theor. p_{th} and comp. p_b pressures, first pulsation, film RB3

Some comments can be made on the comparison between the theoretical and computed pressures depicted in Figure 6.10. Firstly, both curves follow a similar trend, i.e., fairly constant pressure throughout the evolution apart from the initial and final stages where the pressures are higher. Where the computed pressure p_b is fairly constant, its mean value is approximately 5495 Pa, close to value of the ambient free surface pressure p_a. At no point does the computed pressure fall below the vapour pressure p_v.

Note that the low values of p_{th} may be due to too low a value for p_0 being used in (5.2). Unfortunately this variable is difficult to quantify, being at the very early stages of inception.

The maximum value of p_b in Figure 6.10 is 42890 Pa and this occurs towards the end of the first pulsation at time $t = 0.0225$ sec. The ratio of this maximum pressure to the pressure at the inception point p_∞ is 6.16. In their numerical model of the collapse of a vapour/gas bubble attached to a solid wall, Shima and Nakajima (1977) computed p_b/p_∞ to be a maximum of 7 for a bubble similar to the one investigated here. There appears to be good correlation between these two results.

Due to the finite difference methods used in calculating the radial derivatives, some of the initial and final radial data is unavailable for use in computing p_b. Hence it is suspected that the maximum pressure may be even higher than that shown in the figure.

Secondly, we recall that the theoretical pressure was generated by assuming the bubble evolution process to be adiabatic, with the polytropic index $n = k = 1.4$, the ratio of specific heats. Assuming the actual process also to be polytropic, but with n unknown, we write

$$p_b = p_0[V_0/V]^n, \tag{6.5}$$

where V is volume, with the subscript 0 denoting initial quantities. By plotting $\log(p_b/p_0)$ vs $\log(V_0/V)$ for the first pulsation using the computed data the *actual* value of n for the process can be found. We can confidently assume that the process is not adiabatic due to the energy losses from heating discussed earlier in this chapter. In order to examine the possibility that the growth and collapse phases of this pulsation use different thermodynamic processes, it is useful to separate the data for these phases and to find n for each case. Figures 6.11 and 6.12 depict these plots.

The values of n are given by the slope of the linear curve fits, which attempt to find a mean path between the data points. Thus we have $n = 0.402$ for the growth phase and $n = 0.457$ for the collapse. This indicates that both phases are the result of a process falling somewhere between isobaric ($n = 0$) and isothermal ($n = 1$). Since the contents of the bubble are unknown it is

not possible to speculate on whether the processes are isentropic ($n = k$).

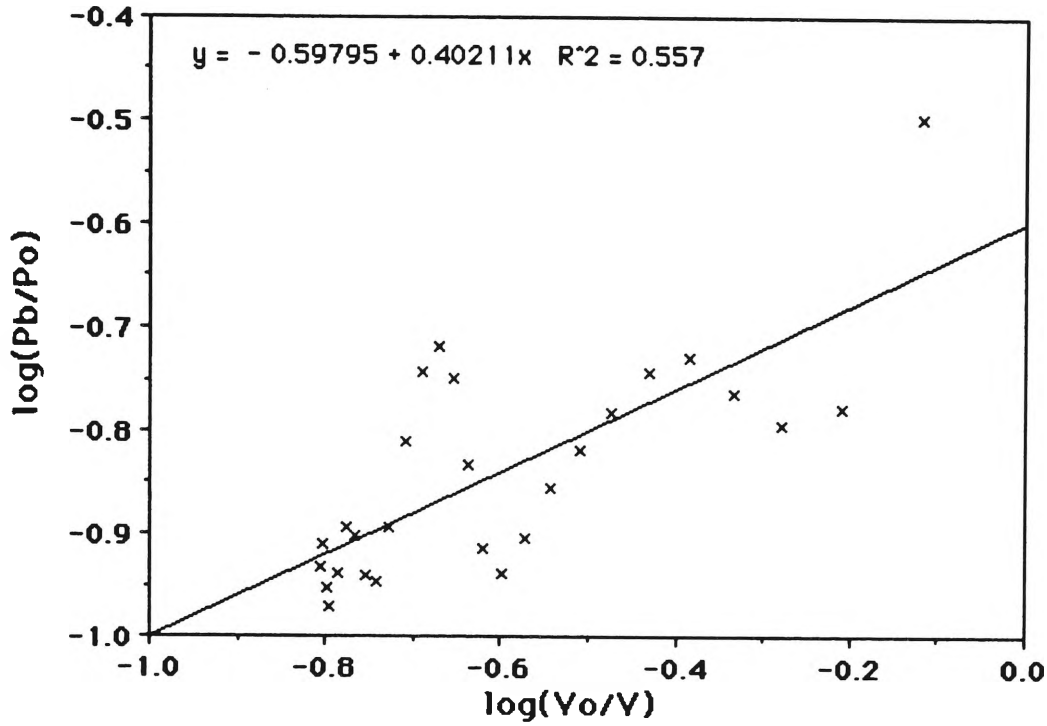

Figure 6.11: 1st growth phase as a polytropic process, film RB3

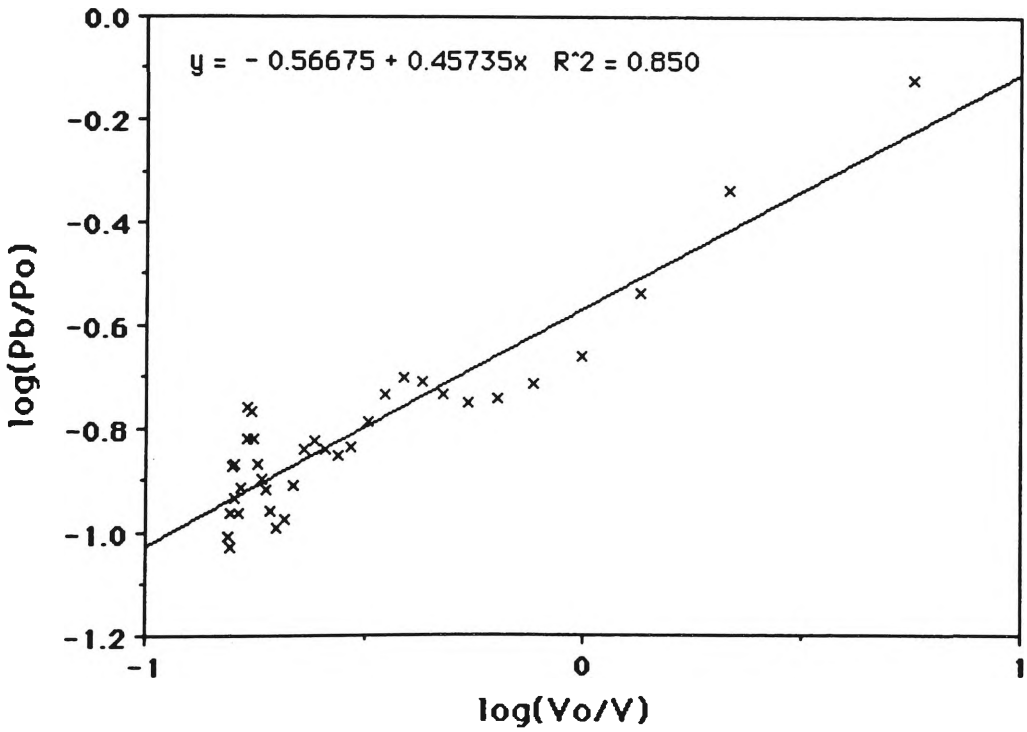

Figure 6.12: 1st collapse phase as a polytropic process, film RB3

Let us now consider the results obtained on a nodewise and timewise basis using the nodal displacement method. A cubic spline representation of the bubble surface was used in preference to linear elements due to difficulties experienced in determining the nodal normal surface velocities $\partial\phi/\partial n$. It seems that the abrupt change in the orientation $\theta_{j-\frac{1}{2}}$ of certain elements at their nodes, combined with the surface geometry, frequently resulted in no solution for the root solving equations (4.44) and (4.47). Hence $\partial\phi/\partial n$ and therefore the pressure p_c was indeterminate there.

Recalling the nodal nomenclature of Figure 4.4, where for the film RB3 the number of processed nodes is 19, the plot of $\partial\phi/\partial n$ for the complete recorded shape history is shown in Figure 6.13. Experimental noise is evident both nodewise and timewise but nevertheless the plot reflects the nodal motion evident in Figure 6.2.

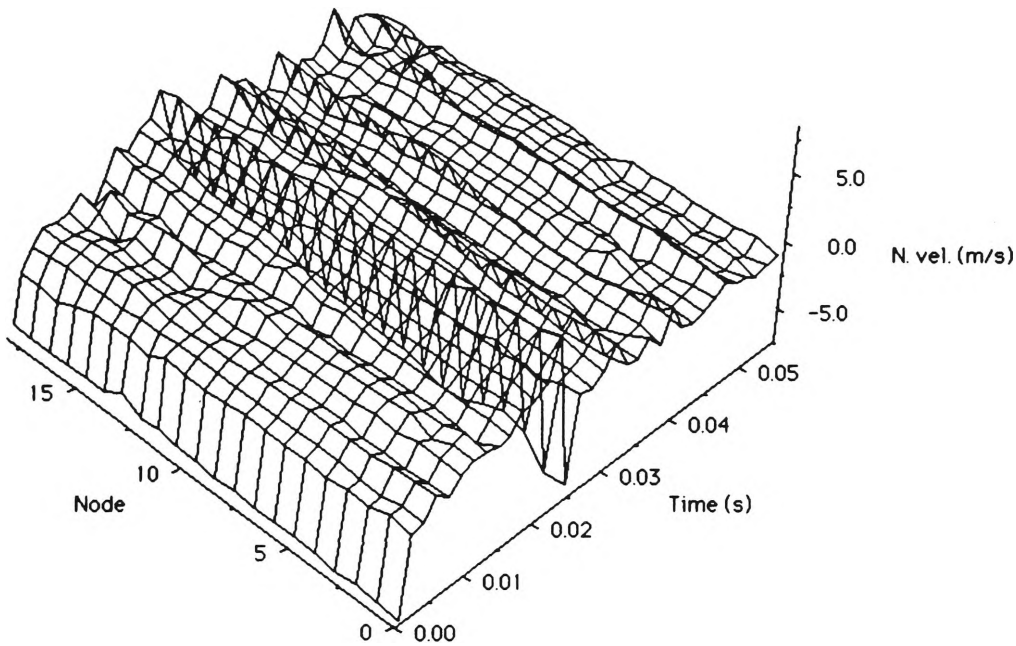

Figure 6.13: $\partial\phi/\partial n$ vs time, complete evolution, film RB3

Since only the first pulsation is of relevance to this work, it is pertinent to display this region enlarged in Figure 6.14.

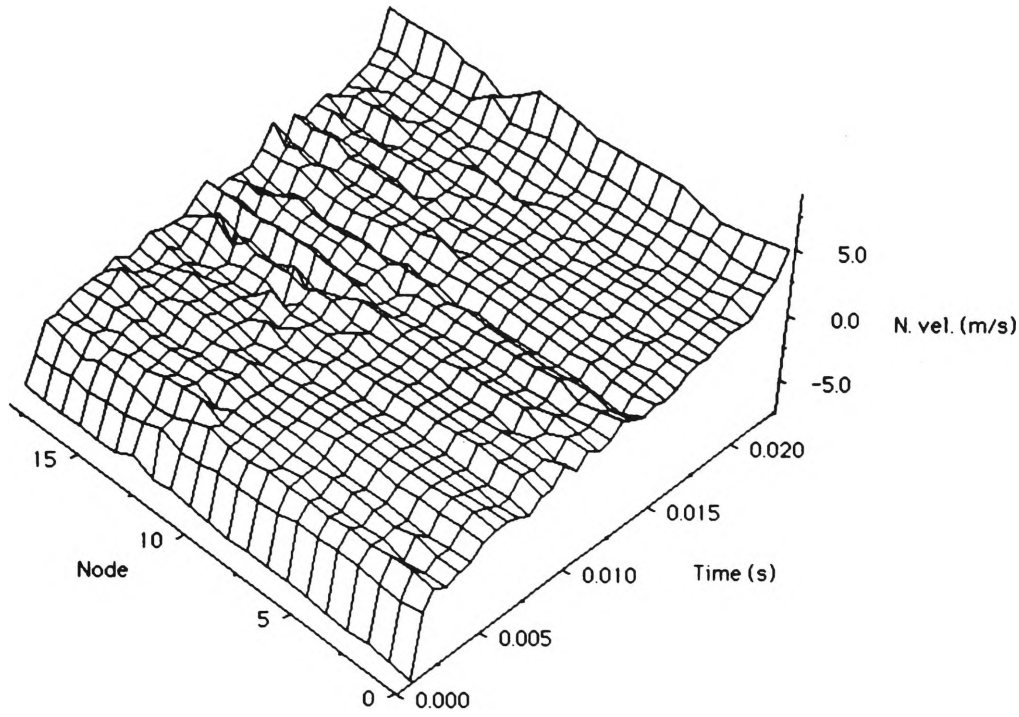

Figure 6.14: $\partial\phi/\partial n$ vs time, 1st pulsation, film RB3

Comparing Figure 6.14 with the plot of $\dot{R}$ in Figure 6.8 it can be seen that both curves follow a similar trend over time. For the duration of the pulsation, the sign of $\dot{R}$ is the opposite of $\partial\phi/\partial n$ at a given time. This can be explained by recalling that $\partial\phi/\partial n$ is measured relative to an *interior* normal whereas $\dot{R}$ is relative to increasing R.

The magnitude of the surface velocities are very similar, both having a maximum value of approximately 7 m/sec. This is significant because it suggests that the tangential surface velocity $\partial\phi/\partial\xi$ is of a small magnitude throughout the first pulsation. That is, because

$$|\dot{R}| \sim |\frac{\partial\phi}{\partial n}|, \tag{6.6}$$

the motion is nearly spherical and therefore $\partial\phi/\partial\xi$ is small.

The nodal fluctuations in normal surface velocity, while significant, do not obscure its nodewise and timewise progression and merely reflect the noise in the images seen in Figure 6.5. The technique for computing the normal surface velocity appears to be able to cope well with the noisy experimental data, and there are no major unexpected deviations from the trend in

increasing $\partial\phi/\partial n$ over time.

The nodewise behaviour of the surface velocity potential ϕ over time is dependent on $\partial\phi/\partial n$ and this dependency can be seen in the nonlinear fluctuations in Figure 6.15.

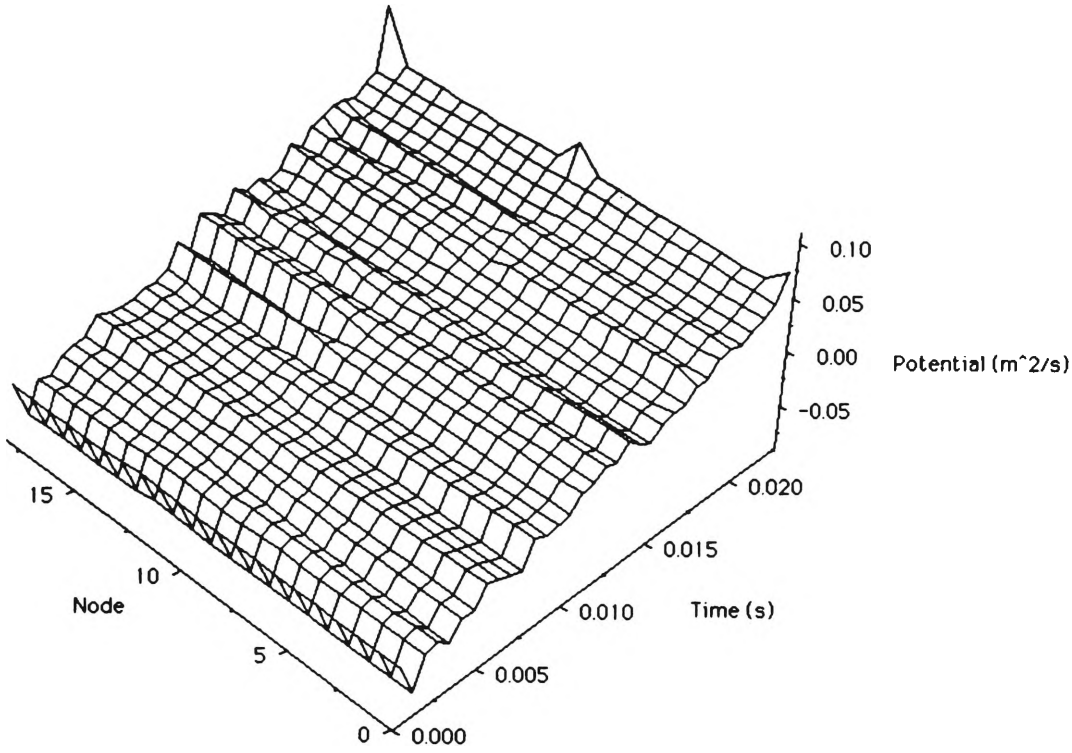

Figure 6.15: Potential ϕ vs time, 1st pulsation, film RB3

The linear trend exhibited by the test data is also seen here, ϕ changing sign about halfway through the first pulsation. Nodewise, there appears to be some improvement in the smoothness of the curve, although spurious peaks are still evident.

Recall that the pressure p_c values are computed using (4.65) which is repeated here

$$p_c = \frac{1}{2}\left[\left(\frac{\partial\phi}{\partial n}\right)^2 + \left(\frac{\partial\phi}{\partial\xi}\right)^2\right] - \frac{D\phi}{Dt} + 1, \tag{6.7}$$

where $D\phi/Dt$ is determined by the trajectory methods discussed in Section 4.3.4. The nodewise and timewise variation in $D\phi/Dt$ and p_c for the first pulsation are depicted in Figures 6.16 and 6.17 respectively.

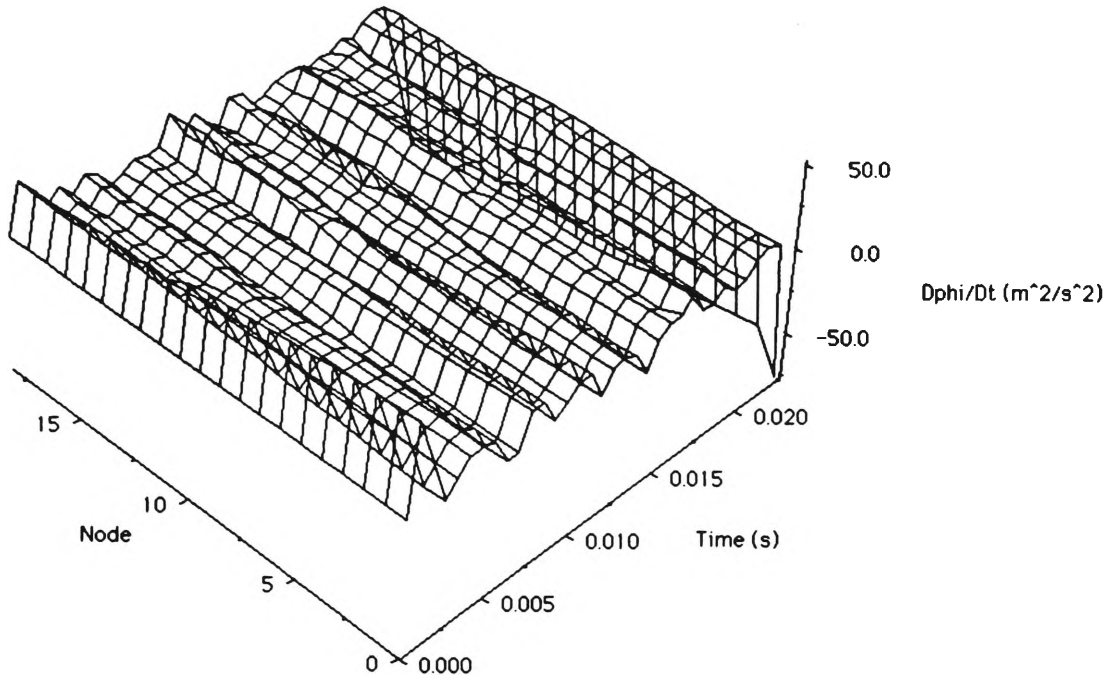

Figure 6.16: $D\phi/Dt$ vs time, 1st pulsation, film RB3

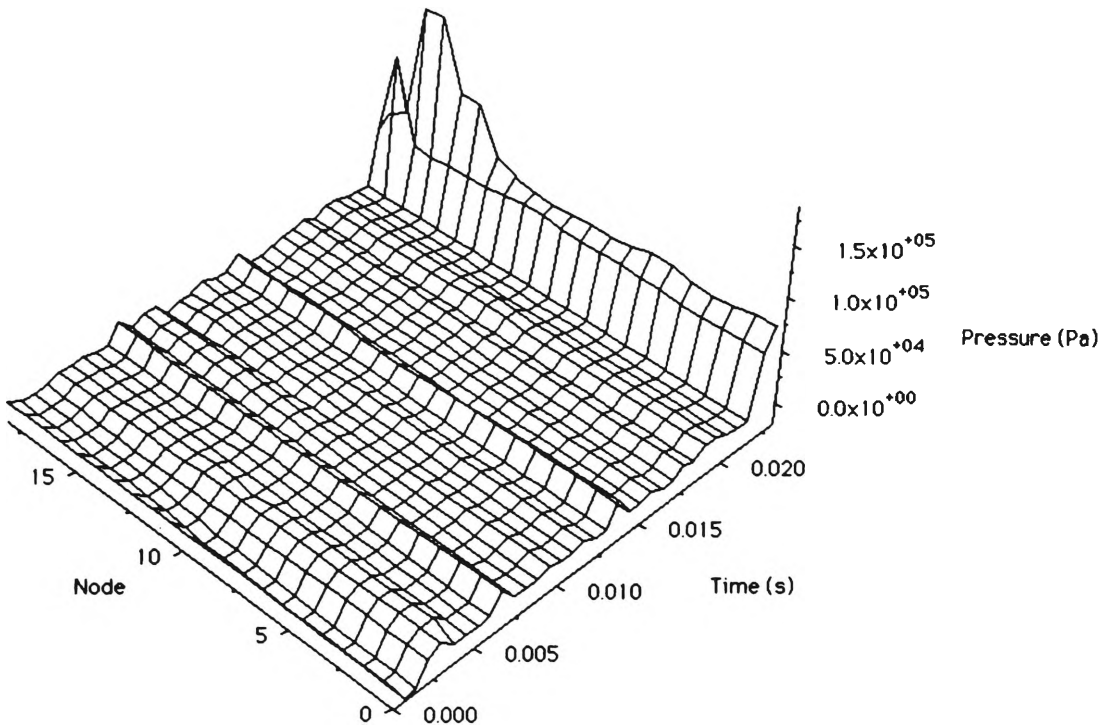

Figure 6.17: Computed pressure p_c, 1st pulsation, film RB3

Noise present in $\partial\phi/\partial n$, ϕ and $D\phi/Dt$ is clearly reflected in the behaviour of p_c in Figure 6.17. Some spurious values are evident in nodes 15 to 18. Although the overall trend in the pressure is difficult to establish, there is a definite ridge visible at the end of the first collapse, around time $t = 0.022$ sec. This is expected because at this point the bubble's volume is a minimum. The value of p_c here averages 80000 Pa, yielding a value of 11.5 for the ratio p_c/p_∞, which is significantly higher than the value of 7 from Shima and Nakajima (1977).

An important observation of theirs, however, is that the surface pressure p_c theoretically varies with nodal position, a result confirmed by the present work. Chahine and Bovis (1983) also found pressure to vary with position on the bubble surface and using the method of matched asymptotic expansions determined that p_c/p_∞ could be as high as 60.

It appears that the average nodal surface pressure may indeed follow a similar trend to the pressure p_b derived using the spherical bubble method in Figure 6.10. Although the early stage of expansion does not reflect this, it is possible that it would if results using the nodal displacement method were available here.

Consider now the appearance of the theoretical pressure p_{th} curve for the first pulsation in Figure 6.18, which was generated using the adiabatic model in (5.2) with the polytropic index $n = k = 1.4$. This model assumes the bubble contents consist of air which exerts a uniform pressure over the surface of the bubble. This nodewise uniformity is clearly visible in the figure.

Some comments regarding the comparison between the experimental and theoretical pressures depicted in Figures 6.17 and 6.18 would be fitting here. Firstly, like the experimental curve, the theoretical curve displays a ridge in pressure at the end of the first collapse. Its magnitude is aproximately one-half of the experimental, at 35000 kPa, and is constant over the surface of the bubble.

Secondly, the nodewise variation in experimental pressure shown in Figure 6.17 is not present in the theoretical curve, showing the possible improvement in accuracy gained by not assuming that the pressure is uniform over the whole of the bubble surface.

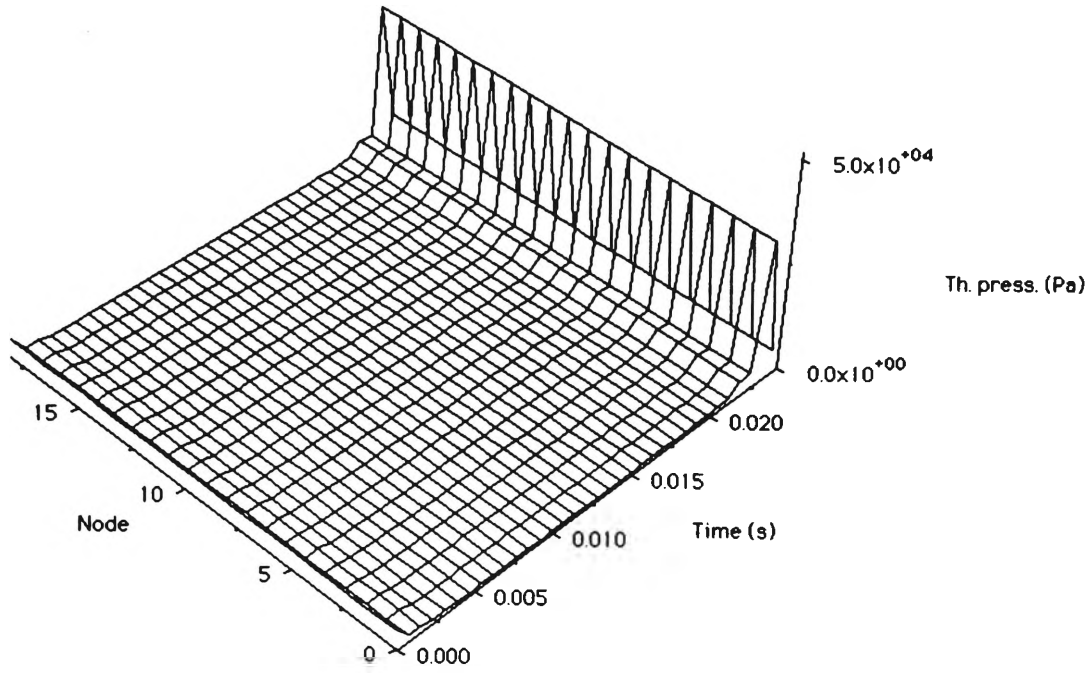

Figure 6.18: Theoretical pressure p_{th}, 1st pulsation, film RB3

Lastly, comparison with the theoretical curve highlights the presence of experimental noise in the raw data. Although there are some defined troughs and ridges in Figure 6.17, it is difficult to attribute these to any part of the bubble evolution process.

Unfortunately due to the degrading influence of the experimental noise, no detailed information can be gained from the pressure results. This by no means depreciates the usefulness of the nodal displacement technique, but does mean that smoother raw data must be used. Films RBS30 and RBS31 will now be examined in this light.

6.2 Film RBS30

The second film that was analysed was RBS30. Whilst the available coordinate data only recorded the first pulsation, this is the most relevant part of the bubble's evolution to this work. More accuracy was afforded in the image analysis process which resulted in an increase in the total number of raw surface nodes *npts* to 72. A summary of the experimental parameters and errors is given in Table 6.3.

Parameter	Symbol	Value
film title		RBS30
date of filming		May 1993
framing rate		5962 /sec
time betweeen frames		167.7μs
analysed timestep	δt	335μs
discharge voltage	V_d	10000 V
discharge capacitance	C_d	1 μF
ambient free surface pressure	p_a	5000 Pa
error in ambient free surface pressure		250 Pa
depth of inception	H	197.5 mm
error in depth of inception		1.0 mm
hydrostatic pressure at inception point	p_∞	6938.0 Pa
error in pressure at inception point		250 Pa
vapour pressure	p_v	1704.0 Pa
error in vapour pressure		55.8 Pa
distance from rigid boundary	h_r	42.5 mm
error in distance from rigid boundary		0.5 mm
bubble orientation		above boundary
water temperature	T_w	15.0°C
error in water temperature		0.5°C
maximum radius	R_m	25.9 mm
error in maximum radius		0.8 mm
standoff parameter	γ	1.638
buoyancy parameter	δ	0.221
frames/first pulsation		132
period of first pulsation		0.0222 sec
scale		0.16120 mm/pixel
error in radius measurement		3 pixels

Table 6.3: Experimental parameters and errors for film RBS30

Once again, every second film image was analysed resulting in 60 Cartesian coordinate files recording the first pulsation of the bubble. A plot of the recorded evolution is shown in Figure 6.19 using the graphic display program in Appendix D. Although it appears that the evolution finishes prematurely, this only because the final film frames in the first pulsation were not digitally analysed. The rigid boundary is represented by the solid line at the bottom of the figure.

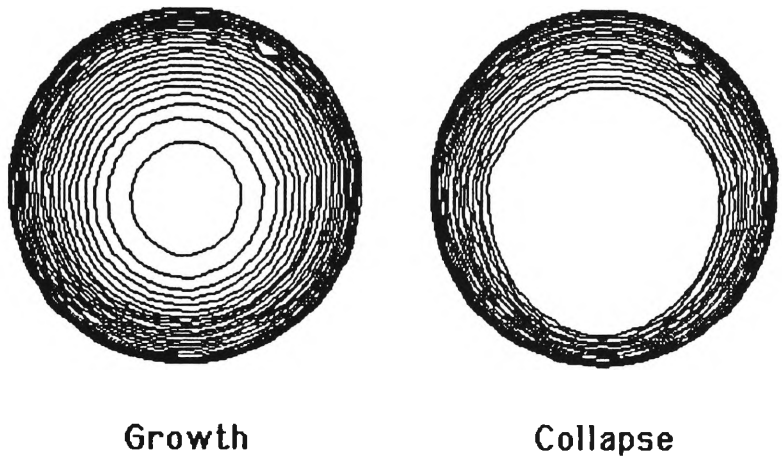

Figure 6.19: Shape history, first pulsation, film RBS30

Inception of the bubble takes place closer to the rigid boundary than for film RB3, and this can be seen to have some effect on its evolution. The Bjerknes force clearly dominates, with the bubble centroid moving toward the rigid boundary in the final stages of collapse. Note that the product $\gamma\delta = 0.362 < 0.442$ so we would expect this translation.

Although some experimental and image analysis noise exists, the bubble adheres to a nearly spherical shape until midway through the first collapse. Thus we would anticipate that the spherical bubble method in particular will yield meaningful results.

The behaviour of the bubble volume V during the first pulsation is shown in Figure 6.20, and this is characterised by a smooth transition over time. The maximum volume, at 0.0000734 m^3, compares favourably with 0.0000643 m^3 for film RB3, and this occurs at time $t = 0.0102$ sec.

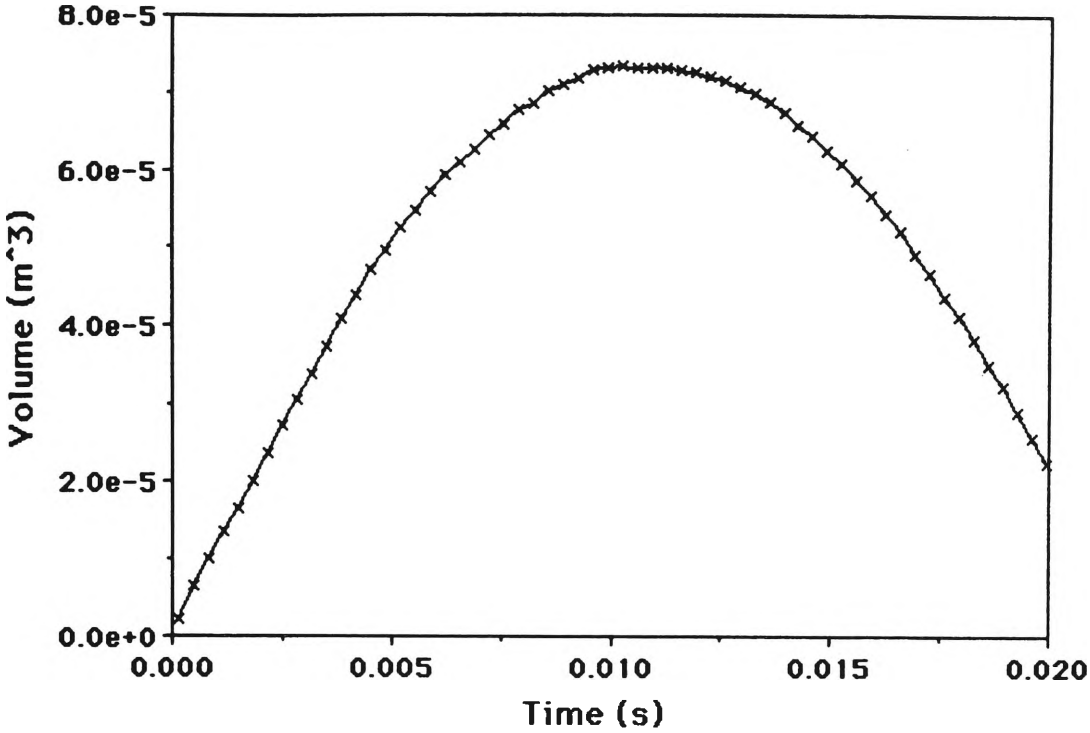

Figure 6.20: Volume V vs time, first pulsation, film RBS30

Figure 6.21 depicts the variation in the kinetic energy E_k of the bubble, computed using (4.98), over time.

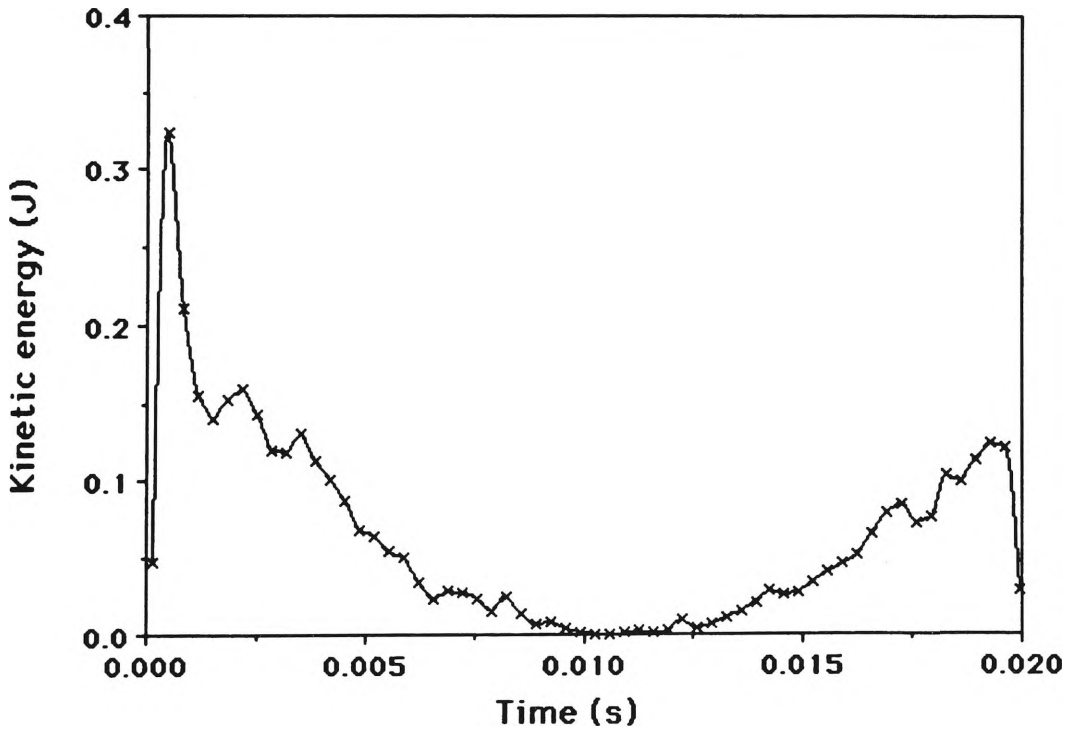

Figure 6.21: Kinetic energy E_k vs time, first pulsation, film RBS30

The initial peak in E_k has a value of 0.324 J, higher than that of film RB3 indicating a more energetic bubble motion. As expected, there is a reduction in kinetic energy over the first pulsation, indicating a pertubation of this motion.

Nondimensional scaling factors generated by the pressure extraction program after the preprocessing of the raw coordinate data are given in Table 6.4.

Parameter	Symbol	Value
no. raw data nodes-1	$npts$	72
no. processed nodes	n	36
length scale	R_m	0.0264
time scale	$R_m(\rho/\Delta p)^{\frac{1}{2}}$	0.01156
velocity scale	$(\Delta p/\rho)^{\frac{1}{2}}$	2.288
pressure scale	Δp	5233.0
volume scale	R_m^3	0.000019
energy scale	$\frac{1}{2}R_m^3\Delta p$	0.0483

Table 6.4: Nondimensional scaling factors for film RBS30

Directly derived from the bubble's volume is the mean radius R and its behaviour is depicted in Figure 6.22 using the smoothing algorithm (6.2). Note the absence of any appreciable noise in the curve. The premature end in the recorded data is evident.

Figure 6.23 shows the mean surface radial velocity $\dot{R}$ curve, which is very similar to that of film RB3, having the same characteristic sign change and linear region. The maximum value of $\dot{R}$ is found to be 5.959 m/sec and this occurs at the beginning of the first pulsation. The small fluctuations in R over time are revealed in the $\dot{R}$ plot, although it does possess greater smoothness than that of RB3.

The derivative of $\dot{R}$, the mean surface radial acceleration $\ddot{R}$, is depicted in Figure 6.24 after smoothing. In this figure the amplifying effect on the fluctuations of $\dot{R}$ can be seen, which are mimicked to some degree in the behaviour of the bubble pressure p_b in Figure 6.25. The theoretical pressure p_{th} is overlaid on this plot, being generated by the adiabatic model of (5.2).

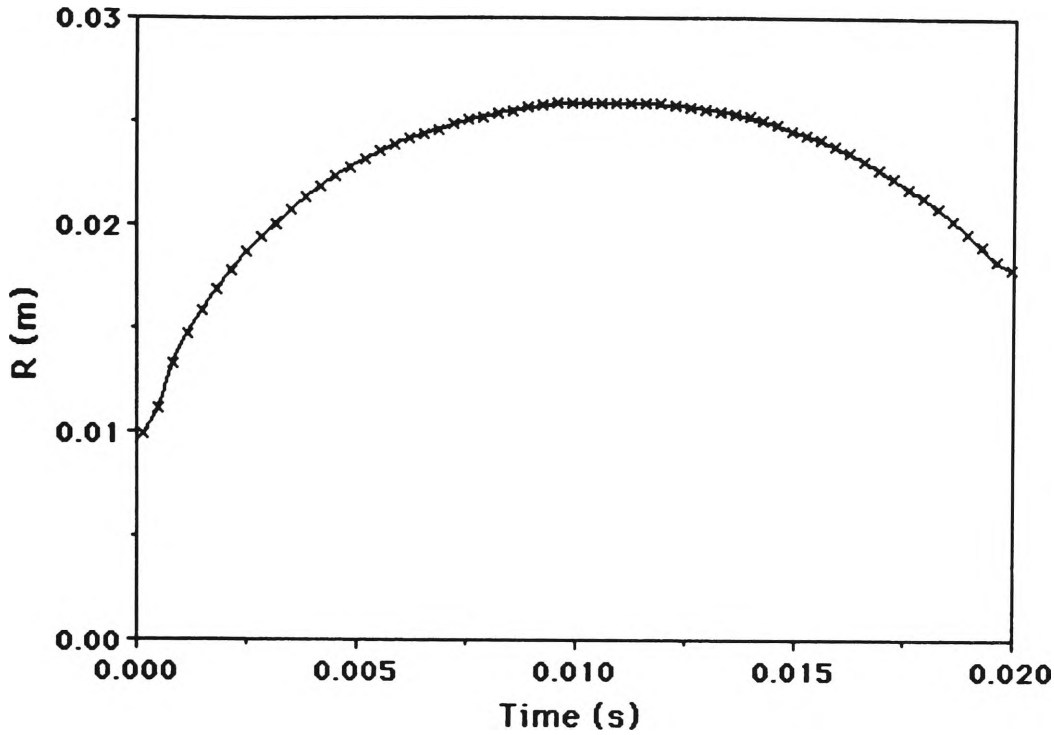

Figure 6.22: Radius R vs time, first pulsation, film RBS30

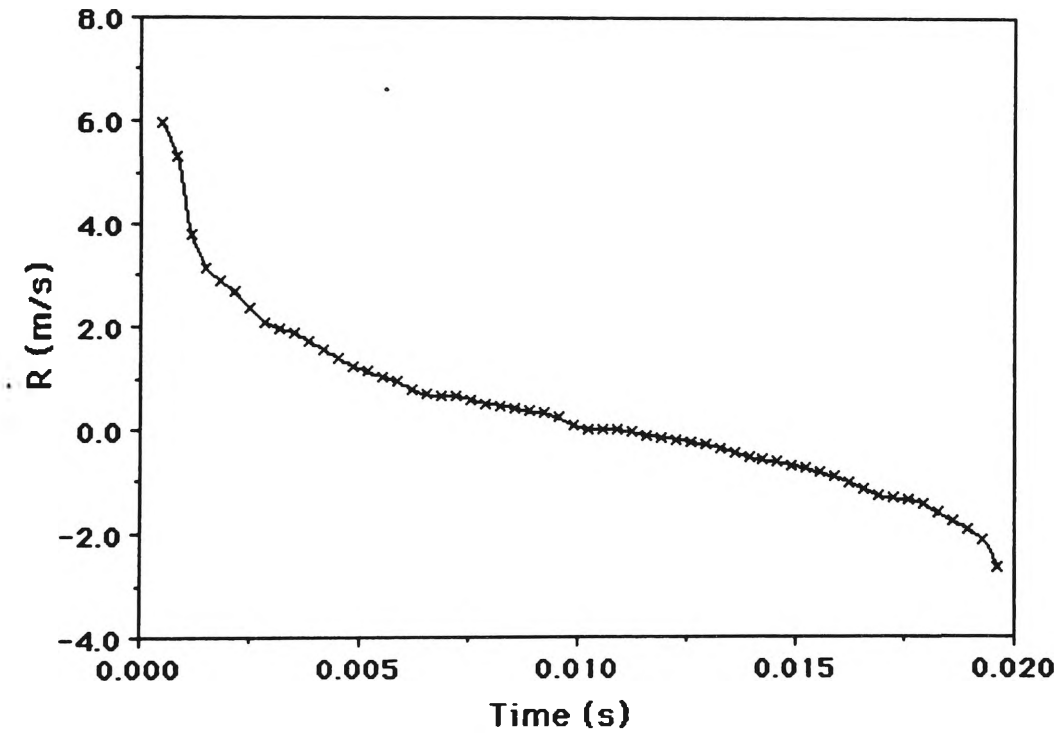

Figure 6.23: Radial velocity $\dot{R}$ vs time, first pulsation, film RBS30

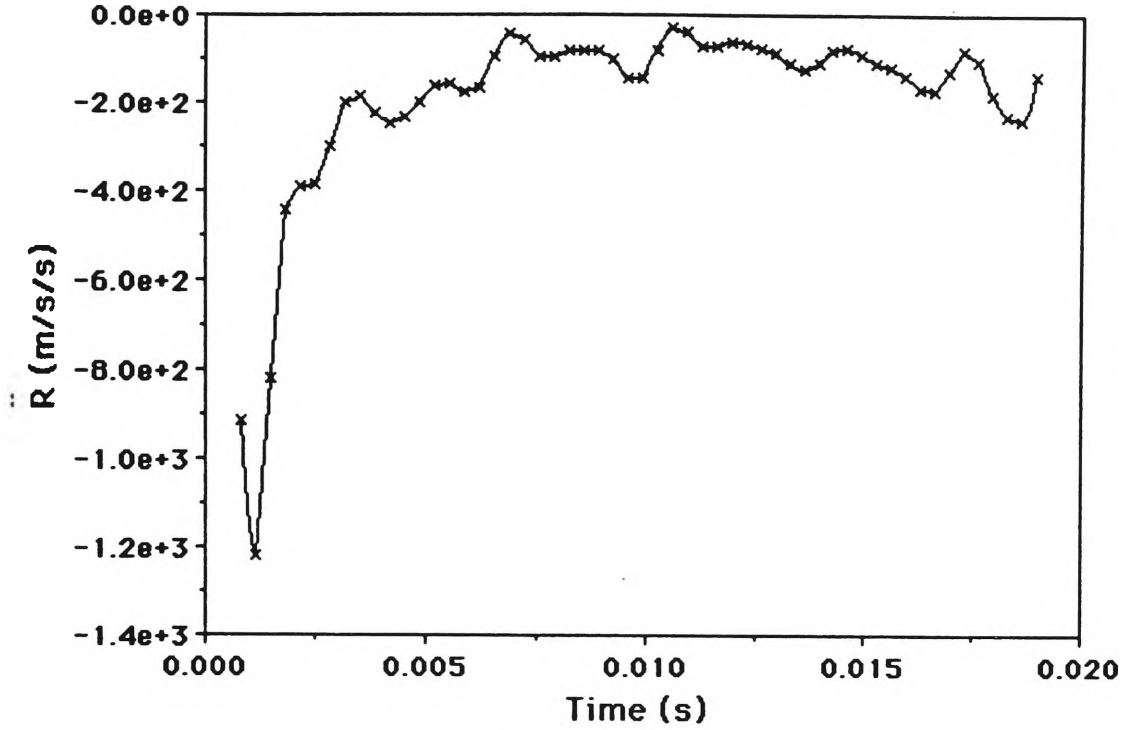

Figure 6.24: Radial acceleration $\ddot{R}$ vs time, first pulsation, film RBS30

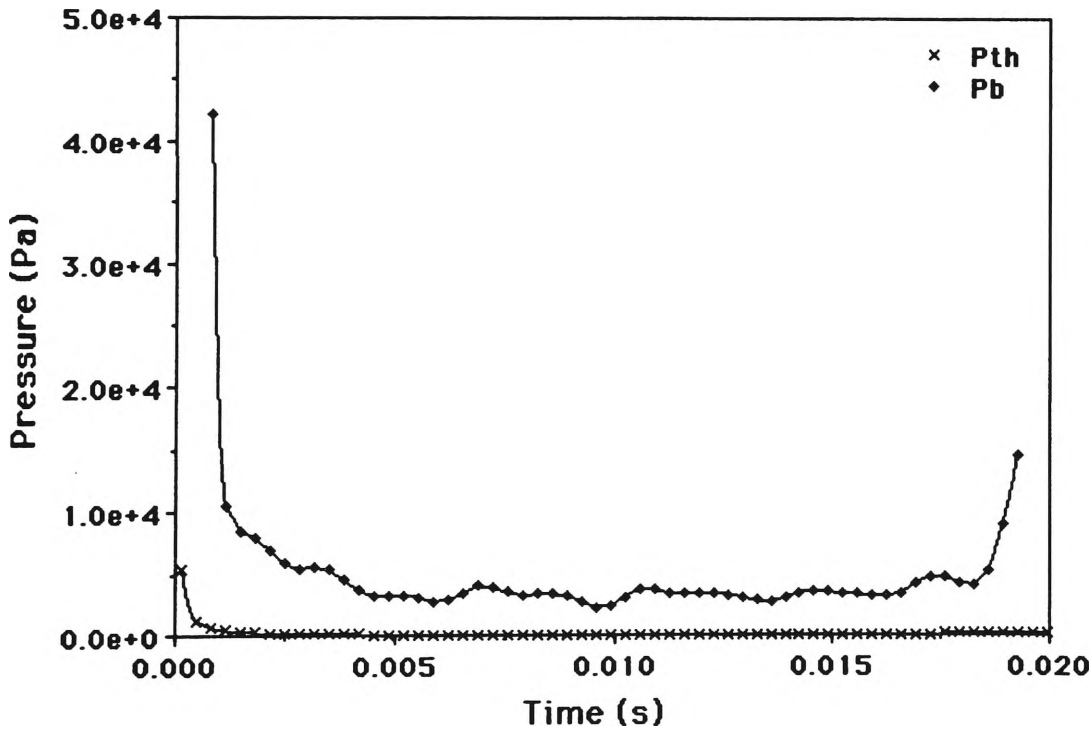

Figure 6.25: Theor. p_{th} and comp. p_b pressures, first pulsation, film RBS30

We see from Figures 6.10 and 6.25 that the pressure history for both films is indeed similar. Once again, the computed bubble pressure p_b for film RBS30 fails to fall below the vapour pressure and has a mean value of approximately 3814 Pa. Due to the premature end to the computed values mentioned earlier we can only speculate as to what the maximum pressure p_b may be. However it is likely to be higher than that shown a short time after inception, at 42200 Pa, which is similar to RB3 for the same time. Like RB3, the generated theoretical pressures p_{th} are much lower than the computed ones, possibly due to a low value of p_0 being used. Due to the lack of data the increase in p_b at the end of the collapse is not shown. The maximum value of the ratio p_b/p_∞ is 6.082, which is similar to Shima and Nakajima's (1977) result.

Let us now examine the bubble evolution thermodynamically. Once again we assume that it is a polytropic process with an unknown index n. By plotting $\log(p_b/p_0)$ against $\log(V_0/V)$ for both the growth and collapse phases we are able to estimate the value of n. Figures 6.26 and 6.27 show these respective plots with their line of best fit. From the figures we have $n = 0.822$ for the growth phase and $n = 0.843$ for the collapse.

Like RB3 this indicates a process somewhere between isobaric and isothermal, but closer to isothermal in this case. Also worthy of note is the excellent correlation for the growth phase, the correlation coefficient being 0.902, which is a good fit indeed. This is a significant result and may well be able to give insight into the thermodynamics of pulsating vapour bubbles.

Progressing to the nodewise and timewise analysis of the data we now examine the results from the nodal displacement method for the film RBS30. Noting that the number of processed nodes has now increased to 37, based on the validation results we would expect more accurate results here than for the previous film. Once again the cubic spline representation of the bubble surface was chosen due to the difficulties experienced with the linear method. Figures 6.28 and 6.29 respectively depict the timewise and nodewise behaviour of the normal surface velocity $\partial\phi/\partial n$ and surface velocity potential ϕ.

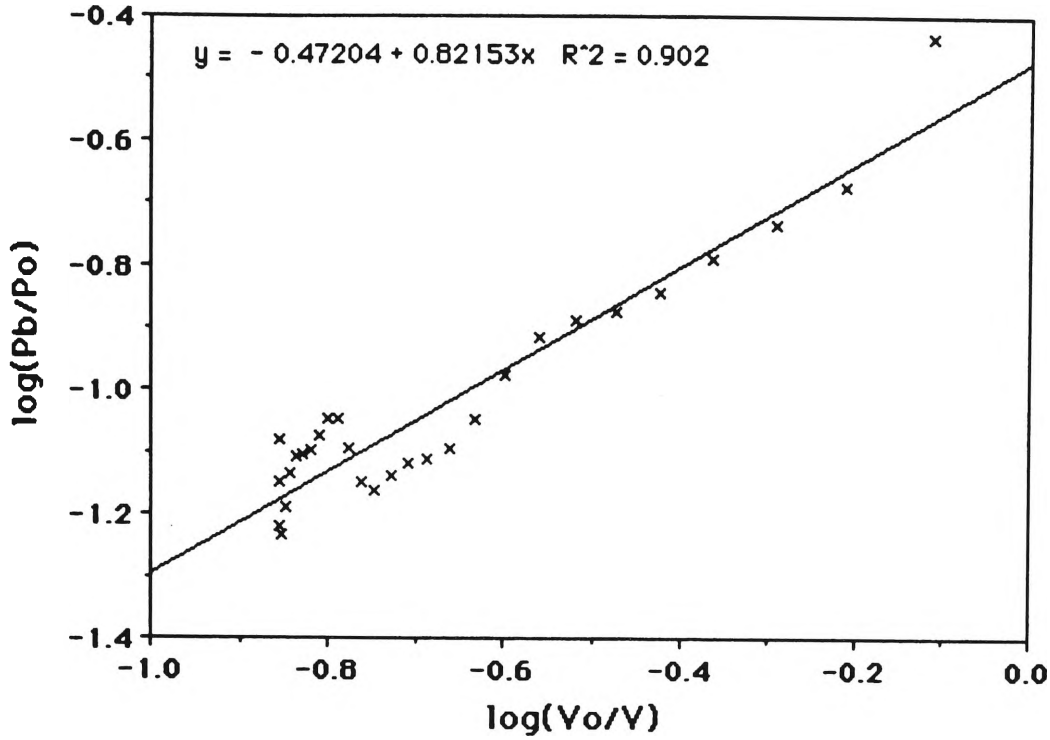

Figure 6.26: 1st growth phase as a polytropic process, film RBS30

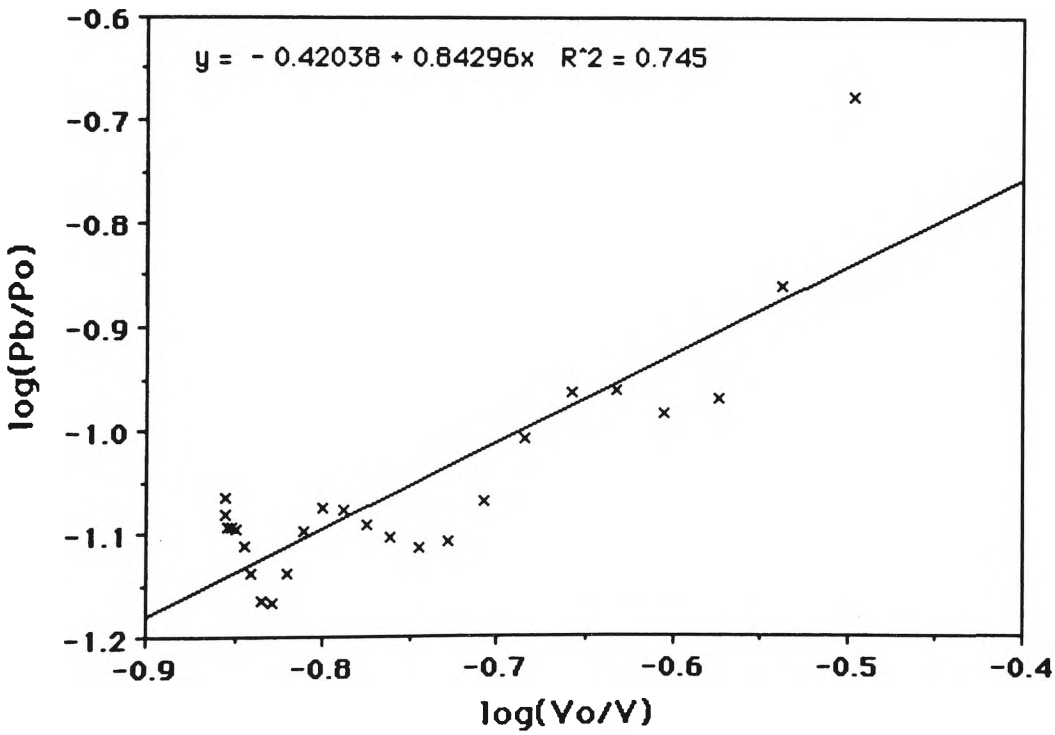

Figure 6.27: 1st collapse phase as a polytropic process, film RBS30

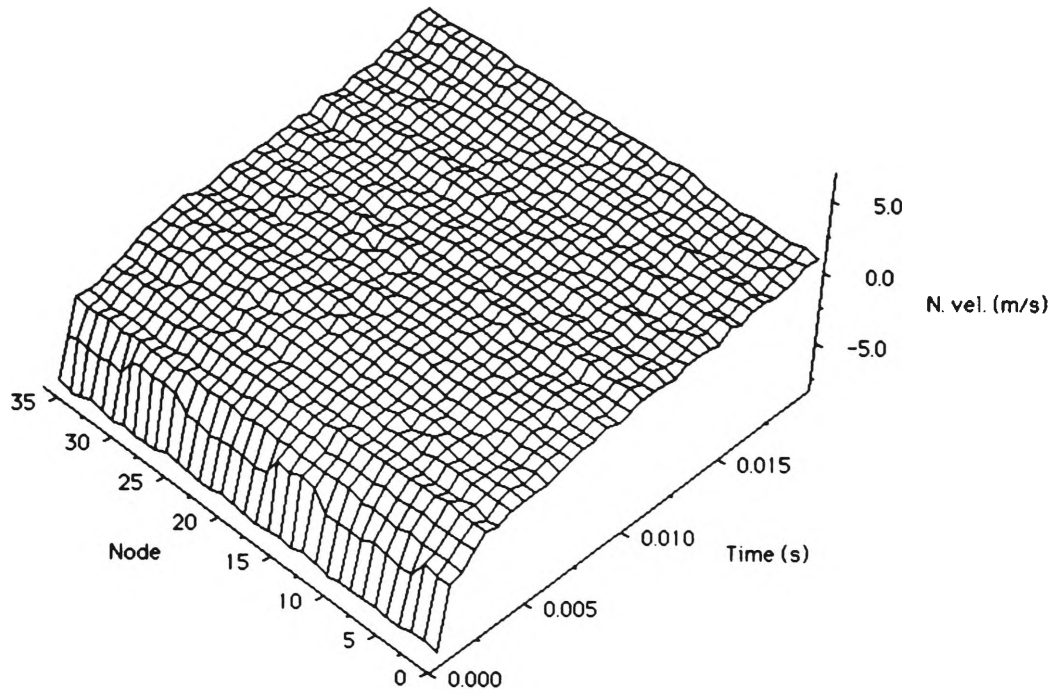

Figure 6.28: $\partial\phi/\partial n$ vs time, 1st pulsation, film RBS30

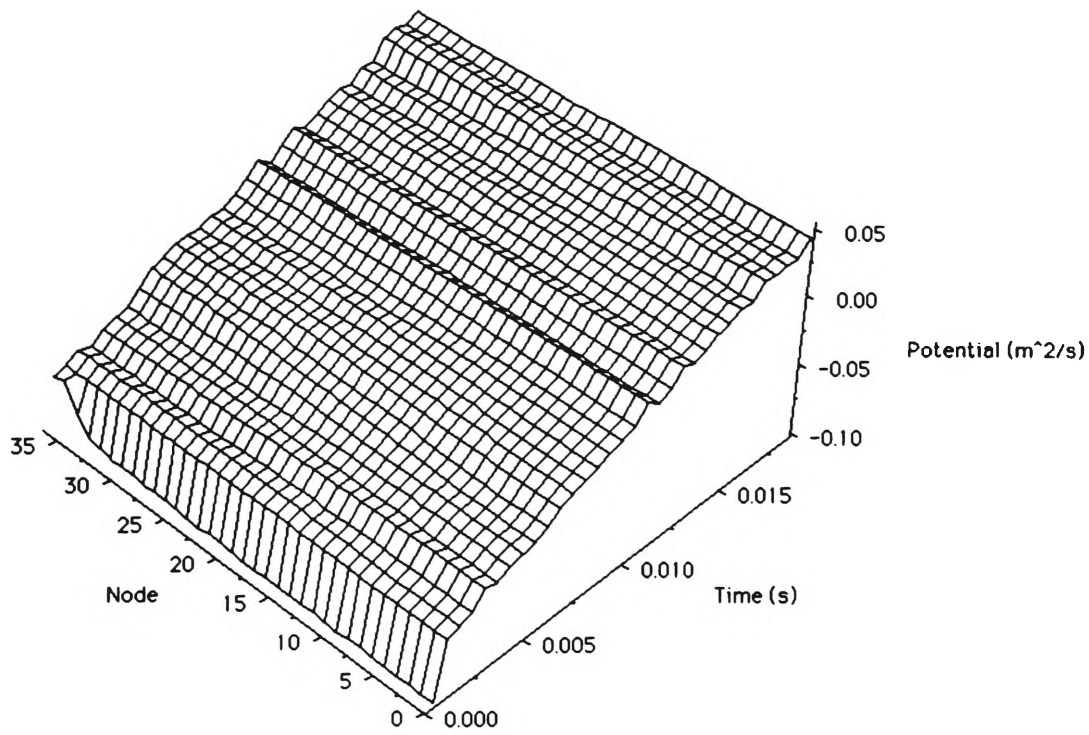

Figure 6.29: Potential ϕ vs time, 1st pulsation, film RBS30

A comparison between Figures 6.14 and 6.28 reveals that the smoother shape history of film RBS30 is reflected in the plot for $\partial\phi/\partial n$. The higher resulution afforded by the increase in the number of nodes allows us to examine the behaviour of this bubble more closely. While there is a nodal variation in $\partial\phi/\partial n$, it is nowhere near as pronounced as that of RB3.

The magnitude of the maximum normal surface velocity is very close to the maximum value of $\dot{R}$ in Figure 6.23, as would be expected. At the end of the first collapse there is a noticeable abscence of the nonlinear behaviour depicted in RB3 for the same point in its evolution. This is due to the premature end of the available data discussed earlier.

The velocity potential ϕ in Figure 6.29 exhibits some fluctuation in value, but this is predominantly timewise. Nodewise, ϕ remains fairly constant, the lack of spurious values indicating the ability of the nodal displacement method to cope well with the smoother data.

Unfortunately this smoothness in ϕ is not reflected in the behaviour of $D\phi/Dt$ seen in Figure 6.30. Here the presence of noise is very pronounced, particularly in the early stages of the evolution. As expected, this directly influences the plot of the computed pressure p_c over time in Figure 6.31, tending to degrade the values so badly that they are practically meaningless.

The lack of coherence in the computed $D\phi/Dt$ and p_c values is very uncharacteristic considering the well behaved values of $\partial\phi/\partial n$ and ϕ both nodewise and timewise. One can only surmise that it is caused by a computational error not evident in the validation process.

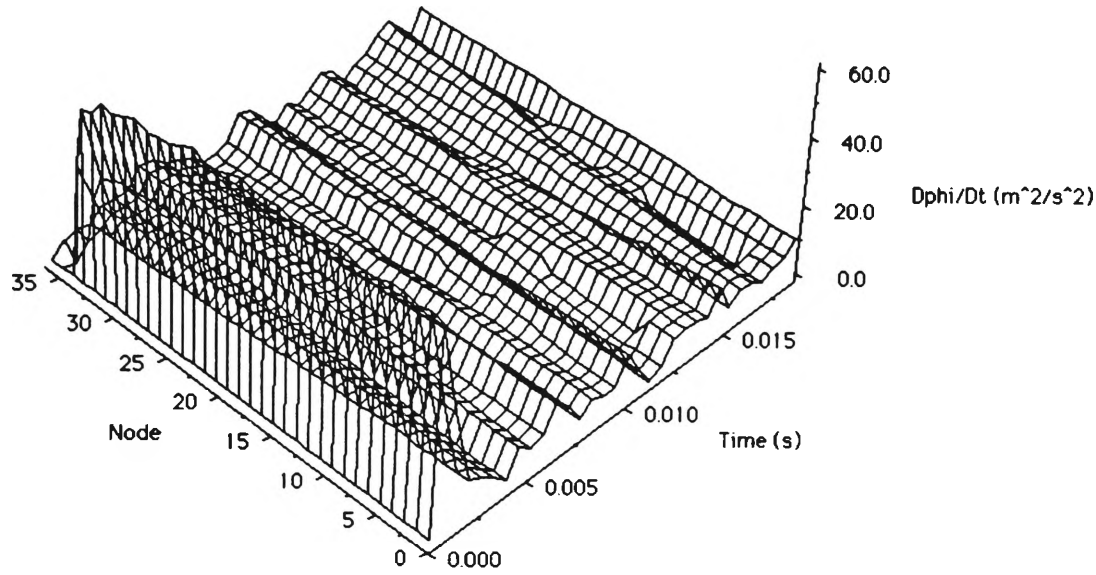

Figure 6.30: $D\phi/Dt$ vs time, 1st pulsation, film RBS30

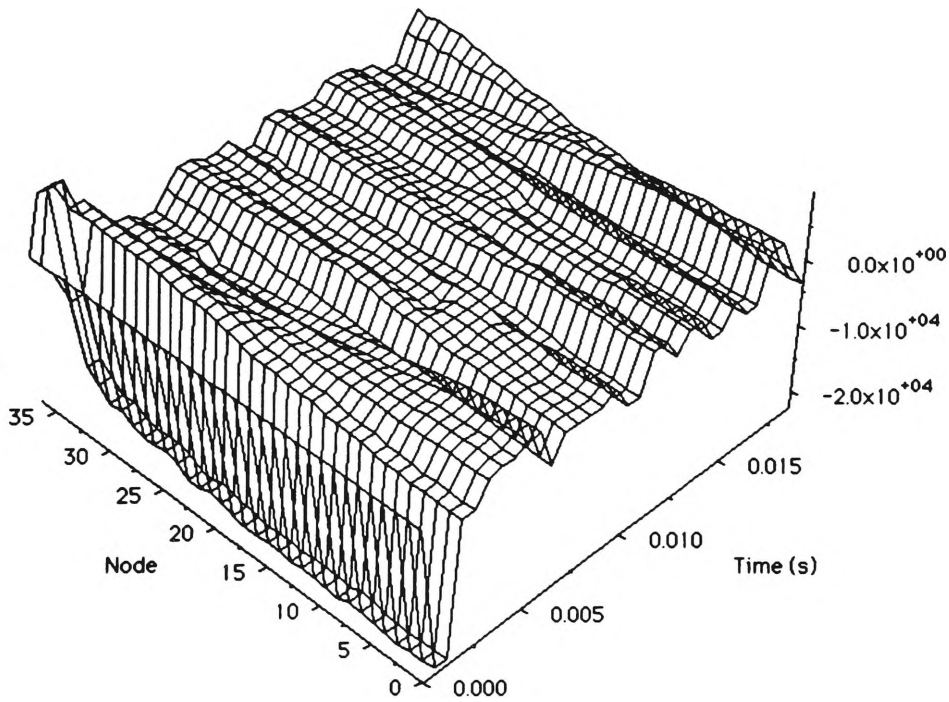

Figure 6.31: Computed pressure p_c, 1st pulsation, film RBS30

6.3 Film RBS31

We now consider the analysis of the third film RBS31. Once again, only the first pulsation was digitised, but unlike RBS30 the collapse was recorded more completely. Table 6.5 summarises the experimental parameters and errors.

Parameter	Symbol	Value
film title		RBS31
date of filming		May 1993
framing rate		5959 /sec
time betweeen frames		$167.8\mu s$
analysed timestep	δt	$336\mu s$
discharge voltage	V_d	10000 V
discharge capacitance	C_d	1 μF
ambient free surface pressure	p_a	5000 Pa
error in ambient free surface pressure		250 Pa
depth of inception	H	204.5 mm
error in depth of inception		1.0 mm
hydrostatic pressure at inception point	p_∞	7006.2 Pa
error in pressure at inception point		250 Pa
vapour pressure	p_v	1704.0 Pa
error in vapour pressure		55.8 Pa
distance from rigid boundary	h_r	35.5 mm
error in distance from rigid boundary		0.5 mm
bubble orientation		above boundary
water temperature	T_w	15.0°C
error in water temperature		0.5°C
maximum radius	R_m	25.7 mm
error in maximum radius		0.8 mm
standoff parameter	γ	1.382
buoyancy parameter	δ	0.218
frames/first pulsation		127
period of first pulsation		0.0214 sec
scale		0.16050 mm/pixel
error in radius measurement		3 pixels

Table 6.5: Experimental parameters and errors for film RBS31

Image analysis of the first pulsation of the bubble yielded 60 Cartesian coordinate files which are plotted in Figure 6.32. The rigid boundary is seen at the bottom of the figure.

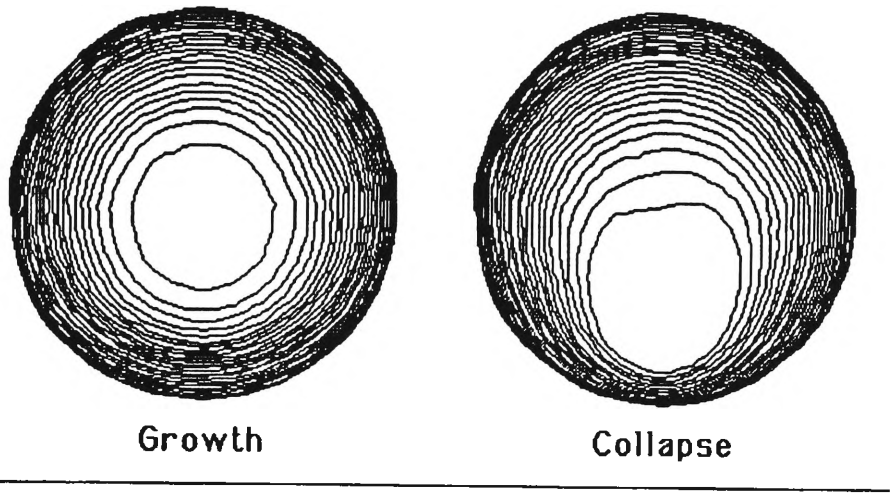

Figure 6.32: Shape history, first pulsation, film RBS31

The strong influence of the rigid boundary is very evident from the shape history of the bubble. Inception takes place closer to the boundary than with the previous two films and so we would expect the Bjerknes force to dominate the bubble motion. This is confirmed by the product $\gamma\delta = 0.301 < 0.442$, a condition which predicts the downward centroidal motion seen in the bubble's shape history.

Note the absence of any significant experimental and image analysis noise in Figure 6.32. We would therefore expect the behaviour of both pressure extraction techniques to be satisfactory in the analysis of this dataset. The spherical bubble method in particular lends itself to the analysis of the first growth phase, where the bubble surface exhibits close adherence to a spherical form. The obvious variation in nodal normal surface velocities during the collapse phase should be easily detected and recorded by the nodal displacement method, and, as will be seen, this is in fact the case.

Over the duration of the first pulsation the volume V displays similar behaviour to that of the previous two films. As can be seen in its plot against time in Figure 6.33 there is however a slight instability near the point of maximum volume, where $V = 0.0000713$ m^3. This is due to the closeness of the consecutive bubble surfaces in the shape history where the

volumes around this stage of the evolution are very similar.

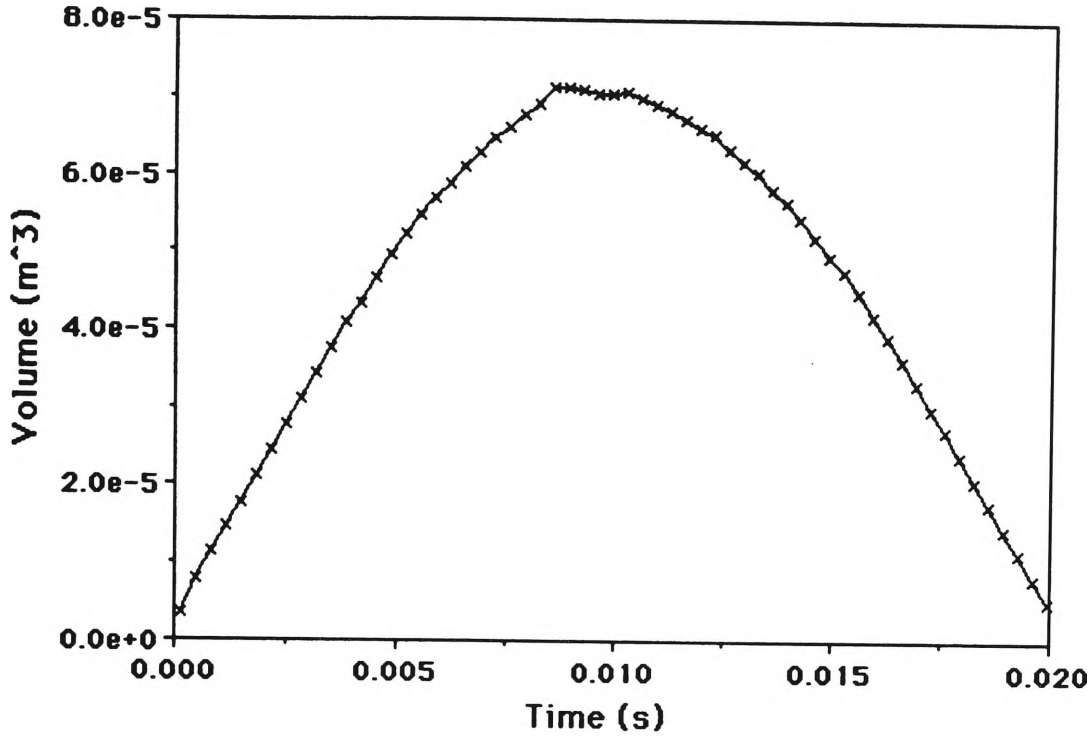

Figure 6.33: Volume V vs time, first pulsation, film RBS31

The change in kinetic energy E_k of the bubble is depicted in Figure 6.34.

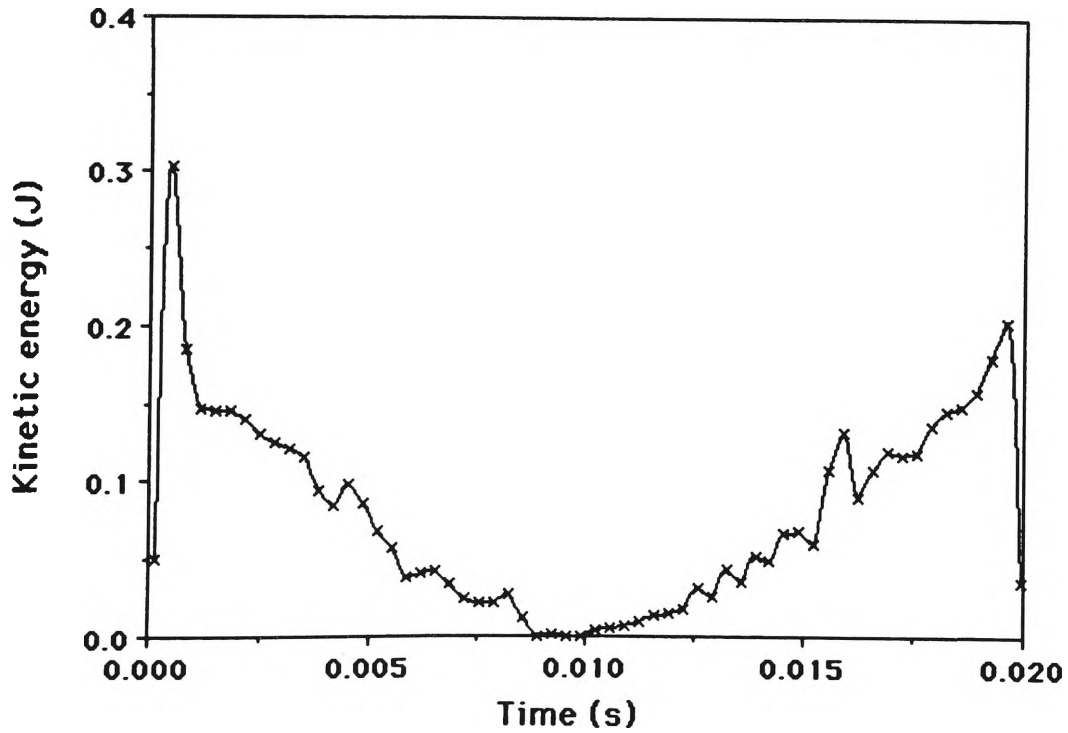

Figure 6.34: Kinetic energy E_k vs time, first pulsation, film RBS31

Following a similar trend to that of the previous two films, E_k peaks just after inception, where it has a maximum magnitude of 0.303 J, and at the end of the first collapse. Once again we see the familiar loss in kinetic energy indicating the pertubation of the bubble motion. Although some noise is present in the results, the improvement in the quality of the raw data can be gauged by comparing the E_k plots of RB3 and RBS31.

We now tabulate the nondimensional scaling factors used in the pressure extraction process in Table 6.6.

Parameter	Symbol	Value
no. raw data nodes	$npts$	72
no. processed nodes-1	n	36
length scale	R_m	0.0261
time scale	$R_m(\rho/\Delta p)^{\frac{1}{2}}$	0.01134
velocity scale	$(\Delta p/\rho)^{\frac{1}{2}}$	2.303
pressure scale	Δp	5302.0
volume scale	R_m^3	0.000018
energy scale	$\frac{1}{2}R_m^3\Delta p$	0.0472

Table 6.6: Nondimensional scaling factors for film RBS31

The behaviour over time of the average bubble radius R is shown in Figure 6.35. Note that unlike RBS30, the complete evolution of the first pulsation is displayed, the curve being very smooth. This is due in part to the application of the smothing algorithm (6.2).

After differentiating R with respect to time, the average surface radial velocity $\dot{R}$ can be obtained, and its behaviour is shown in Figure 6.36. It has a maximum value of 5.991 m/sec just after inception, very close to that of RB3 and RBS30. Any fluctuations in $\dot{R}$ are amplified in the behaviour of the average surface radial acceleration $\ddot{R}$, depicted in Figure 6.37. Whilst the trend in $\ddot{R}$ is similar to that in RBS30, there appears to be an increase in the deviations from the mean values, even though smoothing (6.3) has been applied.

Once again we see that $\ddot{R}$ is the dominating term in the pressure equation (4.13) and its effect on the smoothness of p_b can be seen in Figure 6.38, onto which p_{th}, generated by (5.2), has been overlaid.

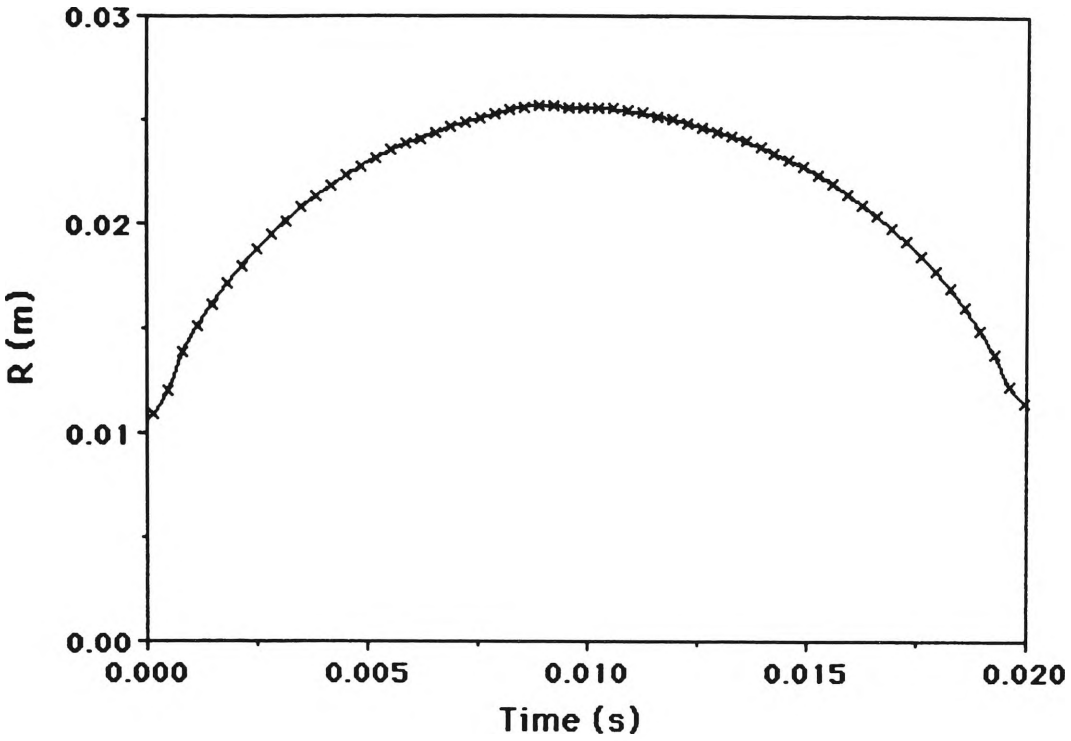

Figure 6.35: Radius R vs time, first pulsation, film RBS31

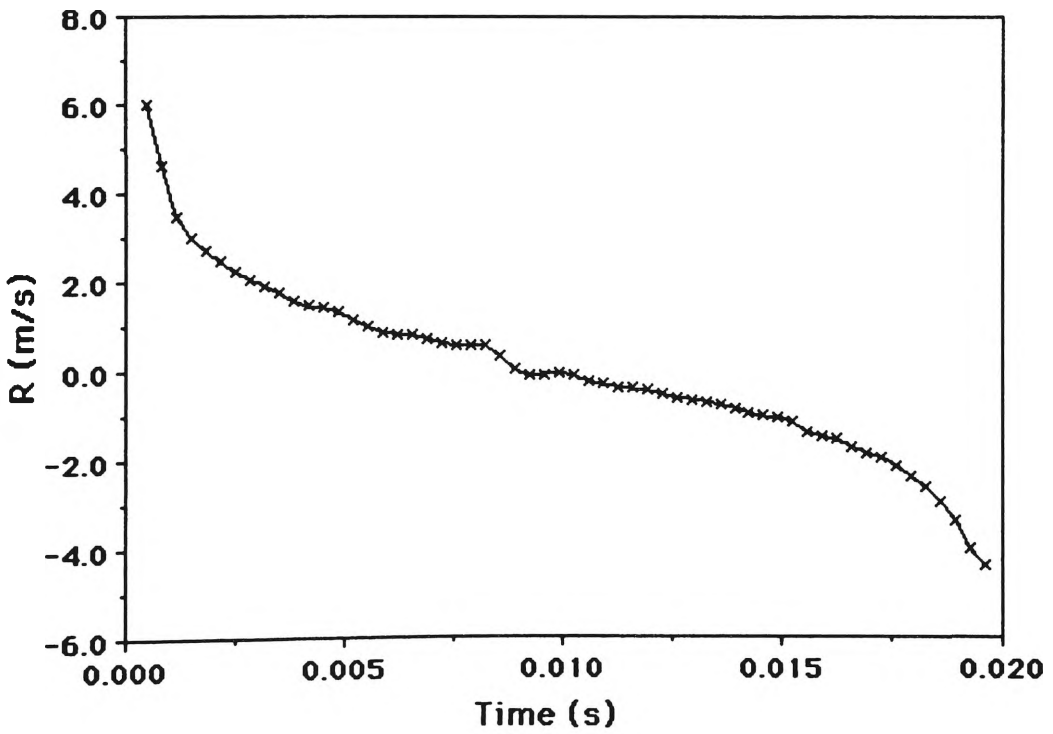

Figure 6.36: Radial velocity $\dot{R}$ vs time, first pulsation, film RBS31

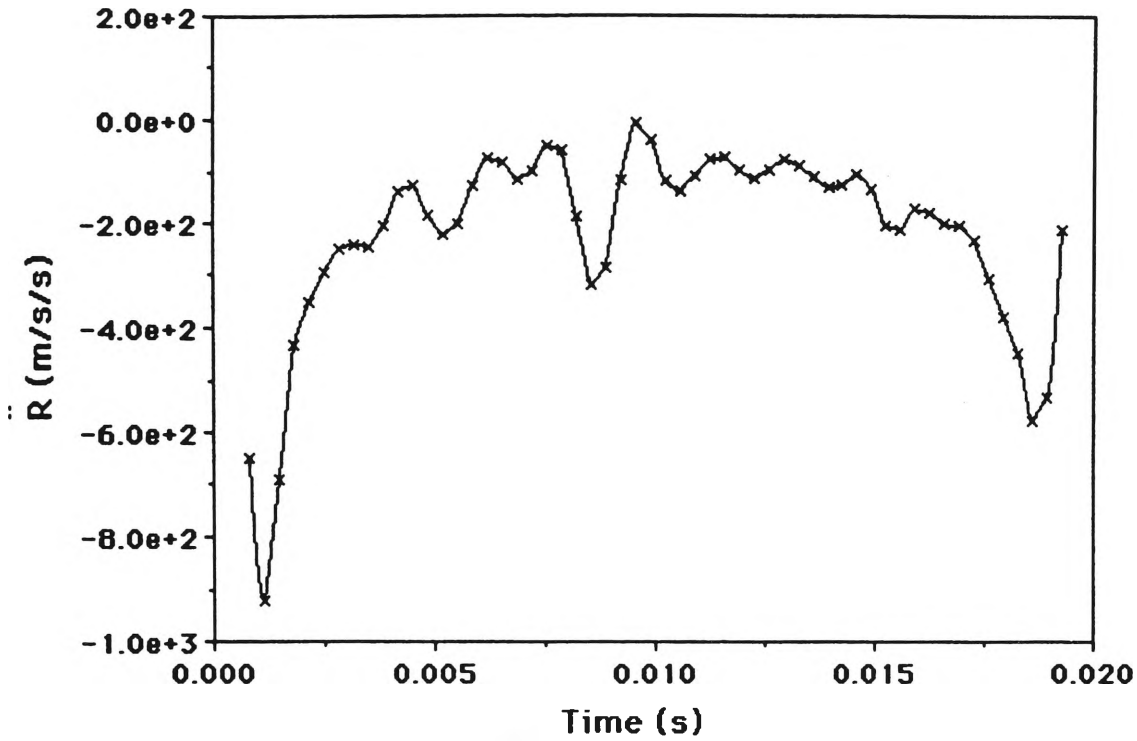

Figure 6.37: Radial acceleration $\ddot{R}$ vs time, first pulsation, film RBS31

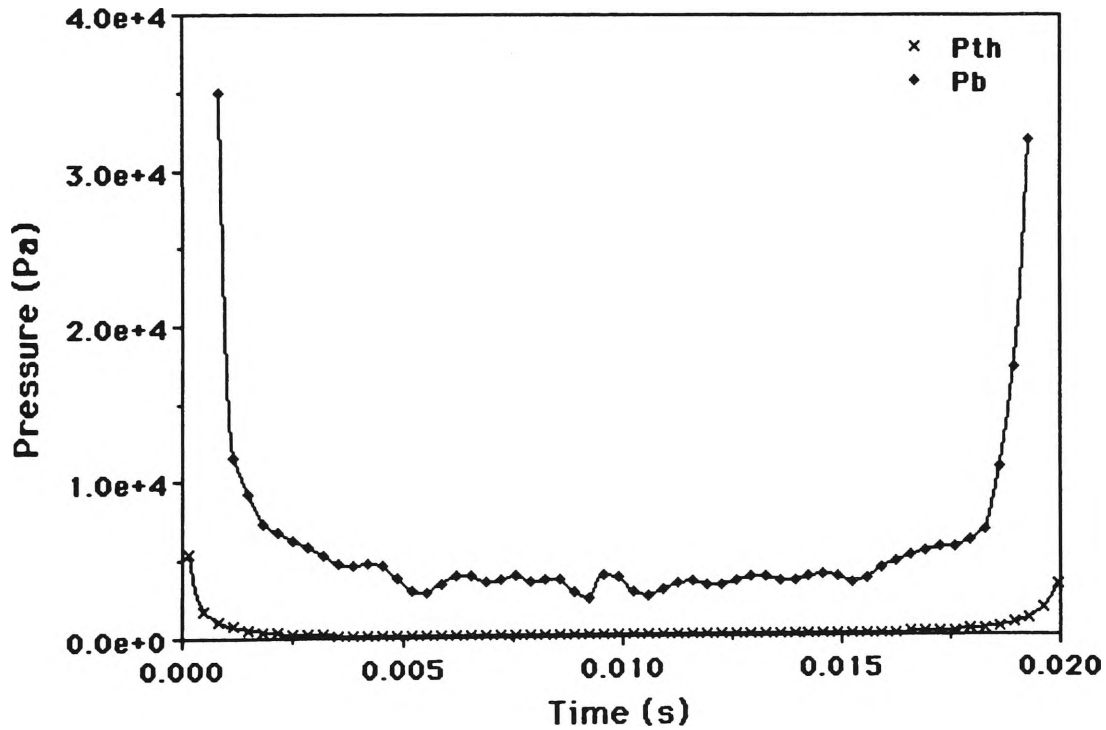

Figure 6.38: Theor. p_{th} and comp. p_b pressures, first pulsation, film RBS31

Similarities between the trends in theoretical p_{th} and computed p_b pressures are evident from Figure 6.38, both having peaks at the beginning and end of the evolution. The mean computed pressure is 4179 Pa and p_b has a maximum of 35025 Pa, lower than for the previous two films. The ratio of the maximum computed pressure to the pressure at the inception point p_∞ is 4.99, significantly lower than Shima and Nakajima's (1977) result.

Promising results are obtained when the evolution is analysed as a thermodynamic process. As with the previous films, the polytropic index n for the growth and collapse phases is found by linearly fitting the plots of $\log(p_b/p_0)$ versus $\log(V_0/V)$. These plots are shown in Figures 6.39 and 6.40 respectively. Note the excellent linear correlation as indicated by the coefficients.

From the figures, for the growth phase $n = 0.809$ while for the collapse $n = 0.848$, suggesting a process somewhere between isobaric and isothermal, but tending to be more isothermal. These results are similar to those of film RBS30, however more confidence is placed in the present results due to the higher value of the correlation coefficient and better quality of the raw data.

Results from the nodal displacement method are now examined, the cubic spline surface representation being retained for reasons discussed earlier. We see from Figure 6.41 that the normal surface velocity $\partial\phi/\partial n$ varies significantly nodewise during the latter stages of the first collapse. This nodewise variation is suggested by the changes in the shape history in Figure 6.32, where the higher value nodes at the top of the bubble undergo larger displacements over time and hence have higher normal surface velocities.

The overall appearance of the $\partial\phi/\partial n$ plot is reasonably smooth and greater accuracy has been obtained by the higher number of surface nodes compared to RB3. Figure 6.42 depicts the timewise and nodewise variation in the surface velocity potential ϕ over the first pulsation, as derived from $\partial\phi/\partial n$. Whilst there are some fluctuations in its smoothness, the overall trend is similar to the previous two films.

Unfortunately this smoothness is not reflected in the plots of $D\phi/Dt$ and pressure p_c in Figures 6.43 and 6.44 respectively, where the uncharacteristic spurosity first seen in RBS30 is again found. This has tended to nullify these pressure results, from which meaningful data is again unable to be extracted.

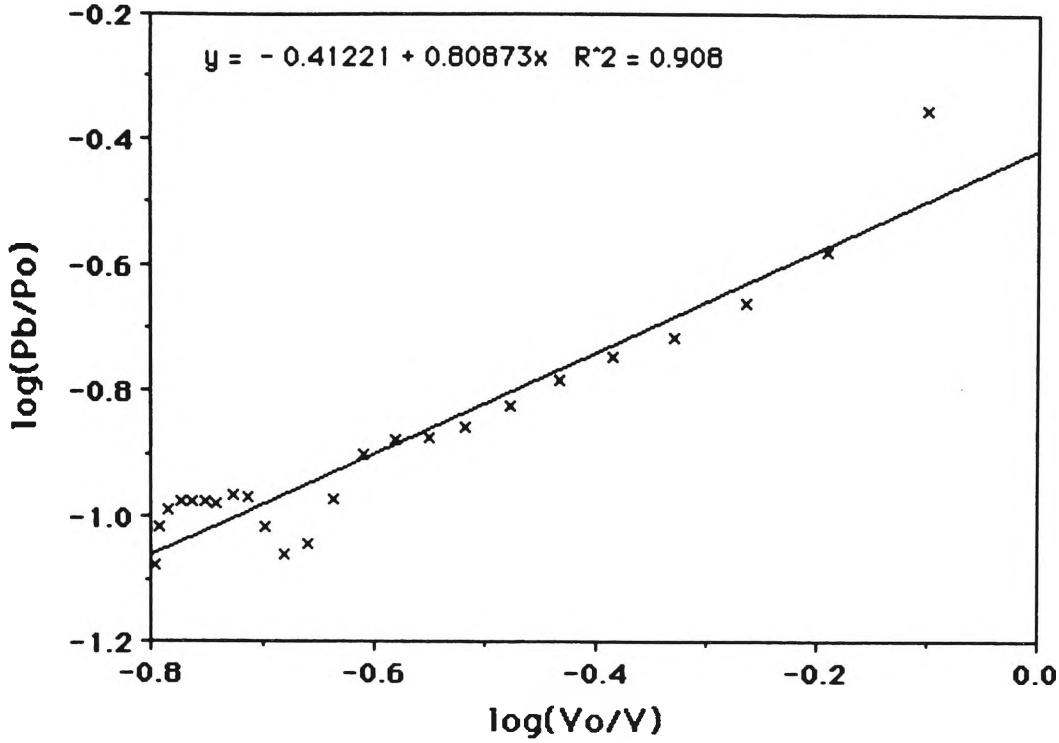

Figure 6.39: 1st growth phase as a polytropic process, film RBS31

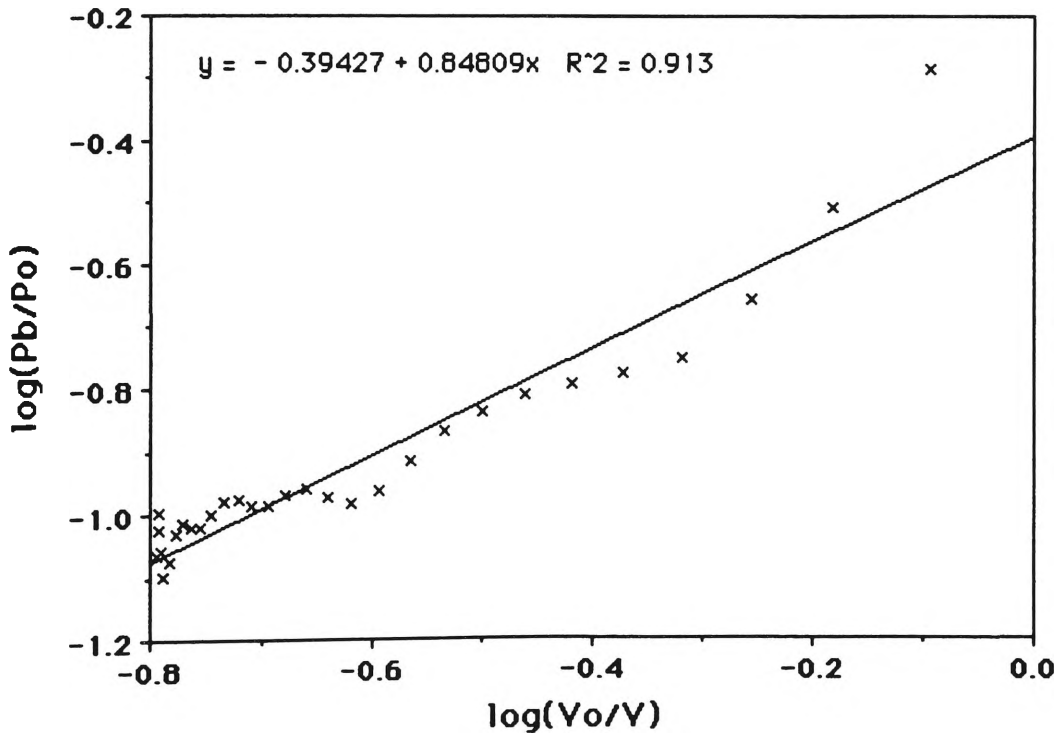

Figure 6.40: 1st collapse phase as a polytropic process, film RBS31

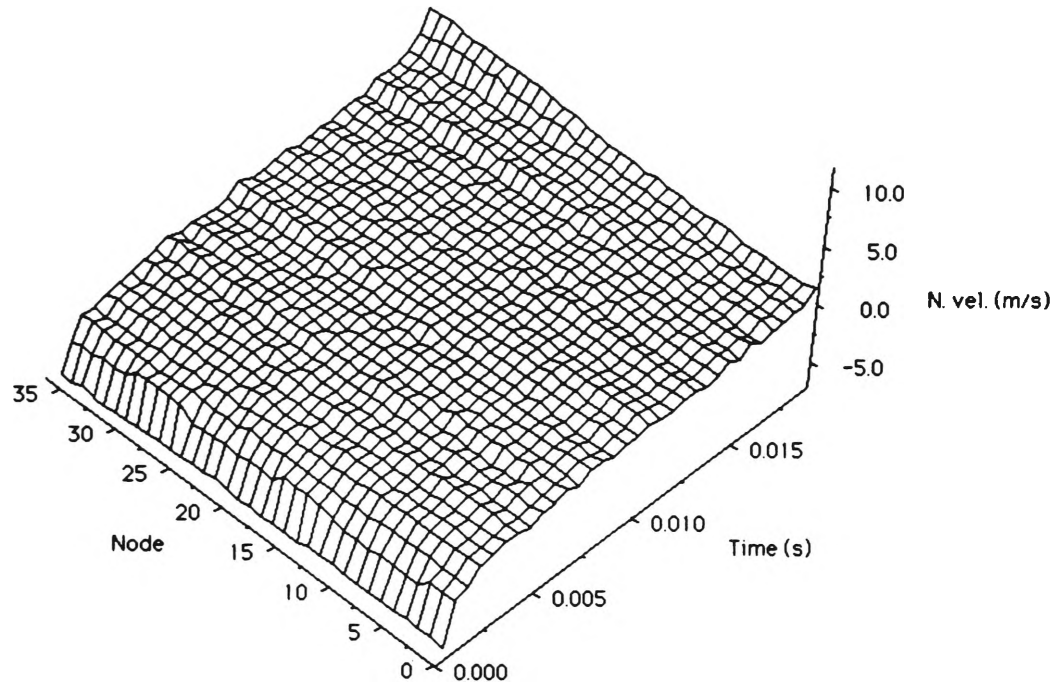

Figure 6.41: $\partial\phi/\partial n$ vs time, 1st pulsation, film RBS31

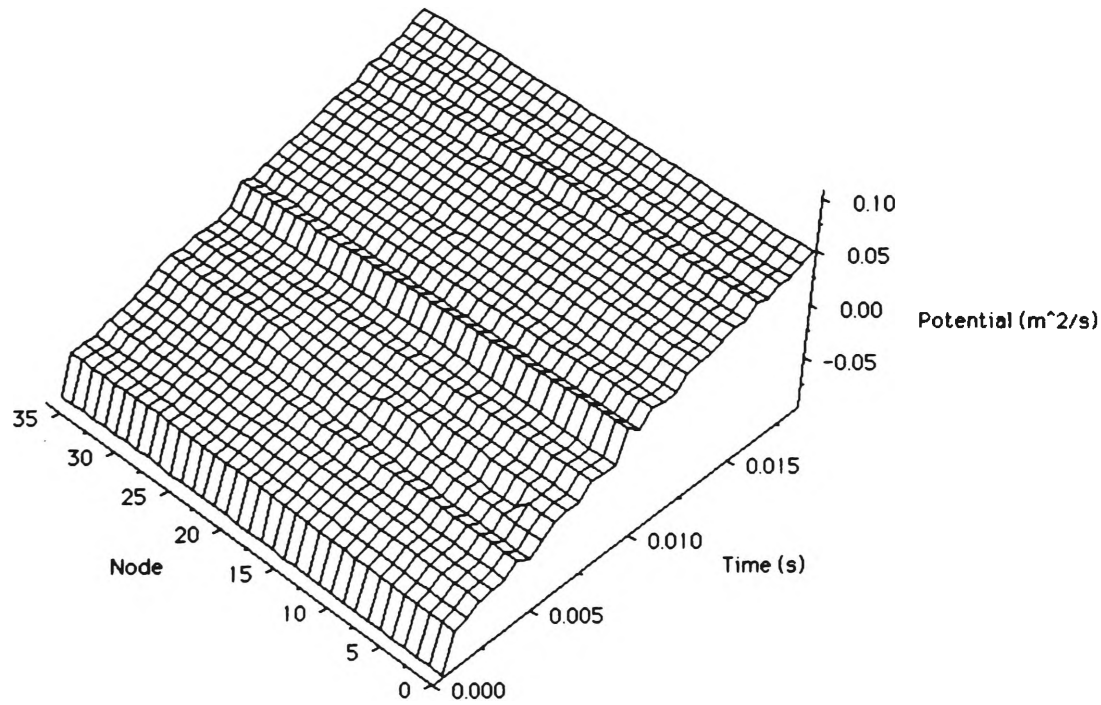

Figure 6.42: Potential ϕ vs time, 1st pulsation, film RBS31

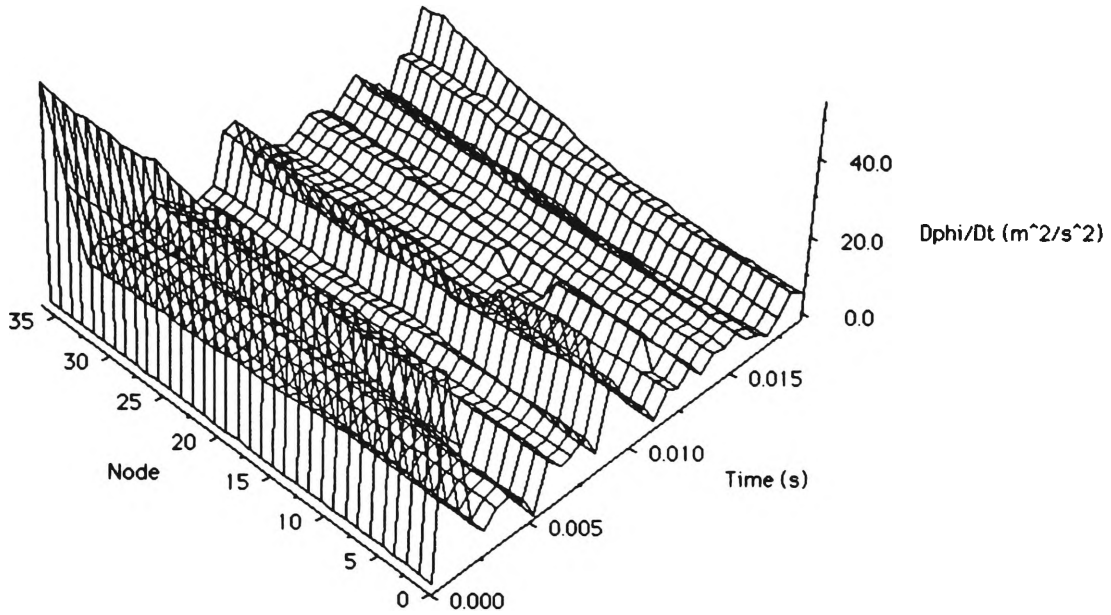

Figure 6.43: $D\phi/Dt$ vs time, 1st pulsation, film RBS31

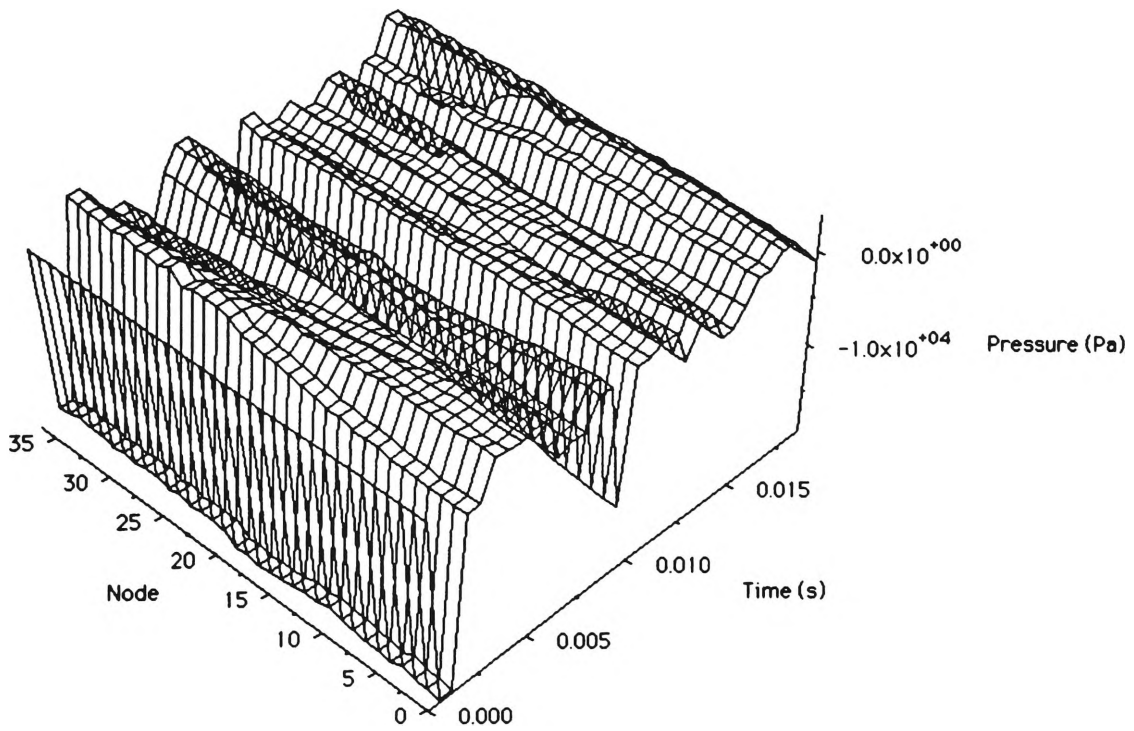

Figure 6.44: Computed pressure p_c, 1st pulsation, film RBS31

Chapter 7

CONCLUSIONS AND RECOMMENDATIONS

The unobtrusive measurement of flow quantities associated with a pulsating vapour bubble has been achieved in this work. The techniques that have been used are unique in the sense that they combine both theoretical and experimental ideas into a single practical engineering tool.

By segmenting the bubble surface, this work has shown that quantities such as normal surface velocity, surface velocity potential and pressure vary along the surface and are not constant as popular theory would suggest. They are dynamic quantities both nodewise and timewise, with greater accuracy in their measurement occurring when the size of these intervals is reduced.

Although not able to yield any meaningful pressure results, the nodal displacement method has nevertheless been shown to be a stable experimental technique for the measurement of normal surface velocity and surface velocity potential. Other gross bubble quantities such as volume and kinetic energy have also been successfully computed. In particular, study of the kinetic energy results has cast some light on the energy losses experienced by a pulsating vapour bubble.

An interesting outcome of this work is that the best results were derived not from the detailed analysis afforded by the nodal displacement method, but rather from the more simpler spherical bubble method, which in fact assumes a uniform pressure inside the bubble. All three films yielded tangible results when analysed in this way, and it would be pertinent to summarise them here.

Firstly, despite a noticeable amount of experimental and image analysis

noise, the smoothing effect of using the average bubble radius as derived from its volume was seen to facilitate the extraction of well behaved radial derivatives. When applied in the pressure equation, these derivatives produced a reasonably stable pressure curve.

Secondly, although the usefulness of this method is restricted to some degree by the ability to record the bubble motion immediately after inception, some idea of the magnitude of the initial bubble pressure can be gained by extrapolating the curve backwards in time. Clearly this would result in extremely high inception pressures, not an unexpected result.

Lastly, and most importantly, by considering the evolution of the bubble as a thermodynamic process, some insight can be gained into just what its characteristic behaviour displays. In most cases it was shown that the bubble expansion and collapse phases bore close resemblance to a process somewhere between isobaric and isothermal, but closer to isothermal. Just what this means with respect to the analysis of bubble dynamics is unclear, and would form the basis of a study in itself.

Some comments regarding the possible directions of any future work will now be given.

Having demonstrated the ability of combined computational/theoretical techniques to extract tangible results from raw experimental data, it would be well worth persevering in this direction in order to obtain even better results. It is recommended that analysis of other shape histories be made and further conclusions be drawn regarding the behaviour of the many measured quantities.

The spherical bubble method, with its inherent data smoothing capabilities, possesses the greatest potential in achieving this end, notwithstanding the requirement that the influence of the rigid boundary be minimal. In this sense this technique is well suited to the analysis of a pulsating vapour bubble in an infinite fluid.

A worthwhile exercise would be to increase the framing rate of the high-speed camera in order to obtain more accurate shape histories. Indeed efforts to accomplish this are currently underway whereby a doubling of the present framing rate is now feasible. Ideally this would allow more meaningful results to be obtained via the nodal displacement method in particular, although the

validation exercise in Chapter 5 failed to show the benefit of such an increase.

Since the digital image analysis of the ciné film is a semi-automatic process, it would not be difficult to increase the number of surface nodes on the bubble. This would be a simple way to improve the accuracy of both methods, as shown in Chapter 5. Of course computing time would increase proportionally although this would be a minor consideration.

The two quantities that appear to dominate the respective pressure results from both methods are $\ddot{R}$ and $D\phi/Dt$. Any improvement in the behaviour of these by means of timewise smoothing would be beneficial. A study of the performance and application of the many known smoothing algorithms with respect to these quantities would therefore be a very worthwhile exercise.

In this work the utility of high-speed photography and digital image analysis has been shown, and exploited fully, without the need for complex physical measurements to be taken.

It is believed that this is a foretaste of things to come in engineering. More and more will image analysis, aided by powerful computing resources, be used as a quantitative engineering tool.

REFERENCES

BENJAMIN, T. B., ELLIS, A. T. 1966 The collapse of cavitation bubbles and the pressures thereby produced against solid boundaries. *Phil. Trans. R. Soc. London Ser. A* **260**: 221–40

BEST J. P. 1991a The dynamics of underwater explosions. Ph. D. thesis, University of Wollongong, Australia.

BEST J. P. 1991b A FORTRAN subroutine for solving a Fredholm integral equation of the second kind that arises in axisymmetric free surface flow problems, unpublished, Materials Research Laboratory, Melbourne, Australia.

BEST J. P. 1992 On geometrical shock dynamics code, unpublished, Materials Research Laboratory, Melbourne, Australia.

BEST J. P., BLAKE, J. R. 1993 An estimate of the Kelvin impulse of a transient cavity. *J. Fluid Mech.* To appear.

BLAKE, J. R., TAIB, B. B., DOHERTY, G. 1986 Transient cavities near boundaries. Part I. Rigid Boundary. *J. Fluid Mech.* **170**: 479–97

CHAHINE, G. L., BOVIS, A. G. 1983 Pressure field generated by nonspherical bubble collapse. *Trans. ASME, J. Fluids Eng.* **105** : 356–63

CHAHINE, G. L. 1982 Experimental and asymptotic study of nonspherical bubble collpase. *Appl. Sci. Res.* **38**: 187–97

ELLIS, A. T. 1956 Techniques for pressure pulse measurements and high-speed photography in ultrasonic cavitation. *Cavitation in Hydrodynamics, HMSO London Pap. 8*, pp. 1–32

ELLIS, A. T., STARRETT, J. E. 1983 A study of cavitation bubble dynamics and resultant pressures on adjacent solid boundaries. *Proc. Intl Conf. on Cavitation, 2nd*, paper C190/83. London: I.Mech.E.

GIBSON, D. C. 1972 The pulsation time of spark induced vapor bubbles. *Trans. ASME, J. Basic Eng.* **94**: 248–49

GIBSON, D. C. 1968 Cavitation adjacent to plane boundaries. *Proc. Aust. Conf. Hydraul. and Fluid Mech., 3rd*, pp. 210–14. Sydney: Inst. Eng.

GIBSON, D. C. 1967 The collapse of vapour cavities. Ph. D. thesis, University of Cambridge, United Kingdom.

KLING, C. L., HAMMITT, F. G. 1972 A photographic study of spark-induced cavitation bubble collapse. *Trans. ASME, J. Basic Eng.* **94**: 825-33

KUCERA, A. 1993 A boundary integral method applied to the growth and collapse of bubbles near a rigid boundary. *J. Comp. Phys.* Submitted.

LAUTERBORN, W., BOLLE, H. 1975 Experimental investigations of a cavitation bubble collapse in the neighbourhood of a solid boundary. *J. Fluid Mech.* **72**: 391–99

LAUTERBORN, W. 1982 Cavitation bubble dynamics – new tools for an intricate problem. *Appl. Sci. Res.* **38**: 165–78

LAUTERBORN, W., VOGEL, A. 1984 Modern optical techniques in fluid mechanics. *Ann. Rev. Fluid Mech.* **16**: 223–44

MITCHELL, T. M., HAMMITT, F. G. 1973 Asymmetric cavitation bubble collapse. *Trans. ASME, J. Fluids Eng.* **95**: 29–37

NAUDÉ, C. F., ELLIS, A. T. 1961 On the mechanism of cavitation damage by non-hemispherical cavities in contact with a solid boundary. *Trans. ASME, J. Basic Eng.* **83**: 648–56

PLESSET, M. S., CHAPMAN, R. B. 1971 Collapse of an initially spherical cavity in the neighbourhood of a solid boundary. *J. Fluid Mech.* **47**: 283–290

RAYLEIGH, LORD 1917 On the pressure developed in a liquid during the collapse of a spherical cavity. *Phil. Mag.* **34**: 94–98

SHIMA, A., NAKAJIMA, K. 1977 The collapse of a non-hemispherical bubble attached to a solid wall. *J. Fluid Mech.* **80 Part 2**: 369–91

SHUTLER, N. D., MESLER, R. B. 1965 A photographic study of the dynamics and damage capabilities of bubbles collapsing near solid boundaries. *Trans. ASME, J. Basic Eng.* **87**: 511–17

SINGER, B. G., HARVEY, S. J. 1979 Cavitation damage studies using plasticene. *Int. J. Mech. Sci.* **21**: 409–16

SMITH, R. H., MESLER, R. B. 1972 A photographic study of the effect of an air bubble on the growth and collapse of a vapour bubble near a surface. *Trans. ASME, J. Basic Eng.* **94**: 933–42

TAIB, B. B. 1985 Boundary integral method applied to cavitation bubble dynamics. Ph. D. thesis, University of Wollongong, Australia.

TOMITA, Y., SHIMA, A. 1986 Mechanisms of impulsive pressure generation and damage pit formation by bubble collapse. *J. Fluid Mech.* **169**: 535–64

VOGEL, A., LAUTERBORN, W., TIMM, R. 1989 Optical and acoustic investigations of the dynamics of laser-produced cavitation bubbles near a solid boundary. *J. Fluid Mech.* **206**: 299–338

YU, C. F., SOH, W. K. 1992 A flow visualisation study of underwater explosion. *Proc. Asian. Congr. Fluid Mech., 5th*, pp. 1177–1180. Taejon.

Appendix A

IMAGE ANALYSIS OF CINÉ FILM RECORDS

(by Mr D. Weise and Dr J. P. Best, DSTO)

Image analysis is carried out on the developed ciné film negative that records the transient bubble motion event. The aim of image analysis is to determine the bubble shape profiles throughout the first expansion and collapse. From this data the bubble volume and hence maximum equivalent bubble radius is determined. The process consists of an interactive extraction of the image of the bubble from the background followed by an automatic determination of a polar coordinate record of the bubble shape from this image.

The film is viewed via a video camera and the analog image input to the Kontron IBAS Version 2.0 image analysis system. This system is an integrated package of hardware and software that provides a wide variety of image analysis functions. Each frame is dealt with in succession. The frame grabber converts the analog image into a digital image which is registered in memory and used as input to the various image analysis routines available in the IBAS system. In the first instance, the grey scale is normalised and a median filter applied to smooth the image. Prior to extracting images of the bubble shape, the scaling relationship is determined. On a frame prior to initiation of the event, the distance between the two points where the electrodes emerge from the perspex supporting arms is measured in pixels using a function from the IBAS system. The measured distance between these two points is then used to define the scale with which measurements in pixels may be converted to measurements in physical units. This scale need

only be defined once for each bubble collapse record.

The film is then advanced to that frame at which initiation of the bubble by spark discharge is evident. This occurrence is characterised by the film (negative) being blacked out due to the high intensity light emitted by the spark discharge. Since initiation of the bubble may have occurred anywhere within the time interval between frames, the zero of time for the event is taken to be halfway between this frame and the preceeding one, with an absolute error of $\pm\delta t/2$, with δt the interval between frames. The high intensity light from the spark discharge persists over the first few frames, so that the first frame analysed is usually the third or later in the sequence that records the bubble motion. On each frame the reference marker must be defined. This is so that images from successive frames may be recorded with respect to a common spatial reference. The reference point chosen is the point where the tungsten electrode emerges from its perspex supporting arm. It is estimated that this point is chosen with an accuracy of ± 1 pixel.

The image of the bubble is then separated from the backgound by a process known as segmentation. A greylevel threshold is set, such that removal of those parts of the image with lower greylevels leaves the bubble shape unaltered, as determined by visual inspection. This same greylevel threshold is generally used for subsequent frames from the same event, but may be altered if required. Segmentation leaves a black and white binary image of essentially the bubble, however, in many cases partial images of the electrodes remain, as do images of any small and spurious secondary cavitation bubbles that are formed around the edges of the main bubble. These parts of the remaining image are manually removed by defining a clipping region around the bubble. At this stage, only the image of the bubble remains, and any holes in the image are filled. The holes that are filled arise during segmentation and are due to the varying levels of illumination that are experienced over the bubble as the background illumination is refracted by the bubble itself. IBAS functions allow automatic determination of the centre of mass, area and equivalent radius of the binary image. These data are written to file, as is the binary image of the bubble at this time. This process is repeated for the desired frames of the film so that a record of binary bubble shapes is compiled.

The manual aspect of the image analysis program is completed with a determination of the exact framing rate. When the film is recorded focussed light from a LED registers a mark on the edge of the film with a frequency of 1000 Hz. The distance between these timing marks is determined and converted to units of frames. Multiplication by the timing frequency yields the framing rate. Since the camera controller was set to trigger the event once it reached a framing rate of approximately 6000 Hz, there are close to six frames between timing marks. The number of frames recording the first oscillation was usually between 40 and 70. In view of this, the framing rate was determined as an average over 36 frames. This completes the manual phase of the image analysis program.

The determination of the bubble shape coordinates is performed automatically using a program written in the IBAS command language. The binary image is sampled at angular intervals, to give a polar coordinate representation of the bubble surface. The origin for this polar representation is the centre of mass of the bubble image, as previously determined. Along each radial line, a bisection method is utilised to find the point at which the transition from white to black occurs. This corresponds to the edge of the bubble. The coordinates of the boundary points in pixels are converted to physical measurements using the scaling determined previously. In this way the complete history of the bubble shapes during the growth and collapse is recorded. In order to obtain an estimate of the error involved in such a determination of the bubble shape coordinates, a greyvalue profile was recorded at the edge of the bubble when it was near maximum shape. This process involves defining a narrow rectangular strip that crosses the bubble boundary. The IBAS system allows determination of the greyvalue in this rectangular strip and plots a histogram of this value at intervals of one pixel. The boundary of the bubble is characterised by a transition from light (bubble) to dark (background) and the greylevel threshold set during segmentation falls withing the range of greyvalues recorded in this transition region. The width of this transition region gives and indication of the error involved in such a determination. Typically, the width of the transition region was 6 pixels, so the absolute error assumed in radius measurements was taken as ± 3 pixels. This was converted to a physical measurement using the scale value.

Appendix B

DATA PREPROCESSING PROGRAM

```
c     ****************************************************************
c     *                                                              *
c     *            MECH 955 - MAJOR ME THESIS  (1992)                *
c     *                 Course code 303 / ME(Hons)                   *
c     *                                                              *
c     *                 STEVEN HARVEY    8421248                     *
c     *                                                              *
c     *               Supervised by Dr W. K. Soh                     *
c     *                                                              *
c     ****************************************************************
c     *                        Program PREPRO                        *
c     ****************************************************************
c     *                                                              *
c     * This program pre-processes a set of raw coordinate files     *
c     * into BUBB11-ready input data.                                *
c     *                                                              *
c     * Coordinates are smoothed and transformed from the            *
c     * cartesian to the cylindrical coordinate system, and          *
c     * are scaled to the max. radius.                               *
c     *                                                              *
c     ****************************************************************
```

```
c       Variable list
c       -------------

c       area()    = area array in pixels                          dbl prec
c       crdfli()  = input coordinate filename array               character
c       crdflo()  = output coordinate filename array              character
c       densty    = fluid density                                 dbl prec
c       dr(0:,)   = radial diference from r array                 dbl prec
c       dx        = horizontal component of dr                    dbl prec
c       dy        = vertical component of dr                      dbl prec
c       gamma     = ratio of specific heats                       dbl prec
c       maxfil    = maximum no. of files allowed                  integer
c       maxnod    = maximum no. of nodes allowed - 1              integer
c       narea     = area value dummy variable                     integer
c       n         = dummy variable                                integer
c       nbott     = node number or original bottom node           integer
c       ndatyp    = data type, 1-exp., 2-validification           integer
c       nfltyp    = output file type, 0-long, 1-induv.            integer
c       ndum      = dummy variable                                integer
c       nleft     = node number of original left node             integer
c       nnodes    = total no. of processed nodes - 1              integer
c       npts      = no. of physical nodes                         integer
c       nright    = node number of original right node            integer
c       ntime     = time value dummy variable                     integer
c       ntop      = node number or original top node              integer
c       numfls    = no. of coordinate files to be read in         integer
c       nxi       = horiz. coord. dummy variable                  integer
c       nybolt    = ref. vertical coord. dummy variable           integer
c       nyi       = vertical coord. dummy variable                integer
c       phiacc    = test condition for spurious errors            dbl prec
c       pi        = trigonometric pi                              dbl prec
c       pincep    = hydrostatic fluid pressure at inception       dbl prec
c       pixel     = length conversion factor                      dbl prec
c       pvap      = vapour pressure                               dbl prec
c       r(0:,)    = radius from centroid array                    dbl prec
c       rav       = average radius                                dbl prec
c       rmaxi     = maximum bubble radius                         dbl prec
c       ro(0:,)   = output radial coordinate array                dbl prec
c       smooth    = smoothing factor (larger-more spherical)      dbl prec
c       time()    = time array                                    dbl prec
c       tinit     = time of first photographic frame              dbl prec
c       tmstep    = timestep betwen subsequent coord files        dbl prec
c       xi(0:)    = input horizontal coordinate array             dbl prec
c       xicent    = horizontal centroidal coord.                  dbl prec
```

```
c       ybdata   = filename of reference point file          character
c       ybolt()  = vertical coord. (pixels) of ref. bolt     dbl prec
c       yi(0:)   = input vertical coordinate array           dbl prec
c       yicent   = vertical centroidal coord.                dbl prec
c       zo(0:,)  = output vertical coordinate array          dbl prec

c  Last update 31/7/93  sbh


        program prepro


c  Dimension variables
c  -------------------

        implicit double precision ( a - h , o - z )

        parameter( maxnod = 38 , maxfil = 200 )

        character*11 crdfli( maxfil ) , crdflo( maxfil ) , ybdata

        dimension xi( 0 : maxnod ) ,
     &          yi( 0 : maxnod ) ,
     &          ro( 0 : maxnod , maxfil ) ,
     &          zo( 0 : maxnod , maxfil ) ,
     &          ybolt( maxfil ) ,
     &          r( 0 : maxnod , maxfil ) ,
     &          dr( 0 : maxnod , maxfil ) ,
     &          area( maxfil ) , time( maxfil )


c  Set constants
c  -------------

        rmaxi = 0.0d0
        pi = 3.14159265358979320
        pixel = 0.191571d-3
```

```
c  Read in the input filenames
c  --------------------------

       open( 1 , file = 'crdfls.inp' )

       read( 1 , * ) npts
       read( 1 , * ) numfls
       read( 1 , * ) tmstep
       read( 1 , * ) pincep
       read( 1 , * ) nfltyp
       read( 1 , * ) phiacc
       read( 1 , * ) tinit
       read( 1 , * ) pvap
       read( 1 , * ) densty
       read( 1 , * ) ndatyp
       read( 1 , * ) ybdata
       read( 1 , * ) smooth
       read( 1 , * ) gamma

       do 10 i = 1 , numfls , 1

           read( 1 , 500 ) crdfli( i )

 10    continue

       close( 1 )

       ntop = ( npts + 1 ) / 4
       nbott = ( npts + 1 ) * 3 / 4
       nleft = ( npts + 1 ) * 2 / 4
       nright = 0


c  Read in the values of ybolt for each coordinate file
c  ----------------------------------------------------

       open( 1 , file = ybdata )
       write( * , * ) 'reading: ' , ybdata
       read( 1 , 510 )
```

```
      do 30 i = 1 , numfls , 1
          read( 1 , 540 ) ntime , narea , nybolt
          time( i ) = dble( ntime )
          ybolt( i ) = dble( nybolt )
          area( i ) = dble( narea )
30    continue

      close( 1 )


c  Read in the input coordinates and transform and smooth them
c  -----------------------------------------------------------

      do 40 i = 1 , numfls , 1

          open( 1 , file = crdfli( i ) )
          open( 2 , file = 'rawcrd.dat' )
          write( * , * ) 'Input coordinate file: ' , crdfli( i )
          read( 1 , 510 )

          do 50 j = 0 , npts , 1

              read( 1 , 520 ) n , nxi, nyi
              xi( j ) = dble( nxi )
              yi( j ) = dble( nyi )
              write( 2 , 550 ) time( i ) , time( i ) , 1.0d0 ,
     &                         xi( j ) , yi( j )

50        continue

          xicent = xi( ntop )
          yicent = ybolt( i )


c  Map the rigid boundary at yb onto the x-y plane and flip the
c  -----------------------------------------------------------
c  images vertically to reflect the actual orientation. Set the
c  -----------------------------------------------------------
c  new origin to ( xicent , ybolt ) i.e.,
c  --------------------------------------
```

```
c    ////////////////////
c
c             * ntop                                    * nbott
c                                                           * npts
c     * nleft        * nright(0)  ========>  * nleft        * nright(0)
c                   * npts
c             * nbott                                   * ntop
c
c  +(0,0)                                  /////////+(0,0)//////


            do 60 j = 0 , npts , 1

                xi( j ) = xi( j ) - xicent
                yi( j ) = yicent - yi( j )

 60         continue


c  Smooth the data in the x direction using the average
c  ----------------------------------------------------
c  distances from the axes.
c  ------------------------

c           Horizontal - Bottom Right
c           -------------------------
            do 70 j = nright , ntop - 1 , 1

                xi( j ) = ( abs( xi( j ) )
     &                    + abs( xi( nleft - j ) ) ) / 2.0d0

 70         continue


c           Horizontal - Top Right
c           ----------------------
            do 80 j = nbott + 1 , npts , 1

                xi( j ) = ( abs( xi( j ) )
     &                    + abs( xi( 2 * nbott - j ) ) ) / 2.0d0

 80         continue
```

```
c  Consider now only the RHS of the bubble and renumber
c  ---------------------------------------------------
c  the nodes from nright (bottom) to nleft (top) and change
c  ---------------------------------------------------------
c  to cylindrical coords, i.e.,
c  ----------------------------

c            * nbott                                  * nleft
c                  * npts
c    * nleft        * nright(0)  ========>                    * ntop
c
c            * ntop                                   * nright(0)
c
c   /////////+(0,0)//////                  /////////+(0,0)//////

c           Bottom Right
c           ------------
            do 100 j = nright , ntop , 1

                ro( j , i ) = xi( ntop - j )
                zo( j , i ) = yi( ntop - j )

 100        continue


c           Top Right
c           ---------
            ndum = 5 * ntop
            do 110 j = ntop + 1 , nleft , 1

                ro( j , i ) = xi( ndum - j )
                zo( j , i ) = yi( ndum - j )

 110        continue

            close( 1 )

 40     continue

        close( 2 )
```

```
c  Compute the average radius and smooth the coordinates to
c  ---------------------------------------------------
c  this reference if smooth .ne. 0
c  -------------------------------

        if ( smooth .ne. 0.0d0 ) then

         write( * , * ) 'Additional radial smoothing applied'
         write( * , * ) 'smooth % = ' , smooth * 100.0d0


         do 130 i = 1 , numfls , 1

            rav = ( area( i ) / pi ) ** 0.5

            do 130 j = nright , nleft , 1

                r( j , i ) = ( ro( j , i ) ** 2 + ( zo( j , i ) -
     &                         zo( ntop , i ) ) ** 2 ) ** 0.5
                dr( j , i ) = r( j , i ) - rav
                dx = -1.0d0 * dr( j , i ) * ro( j , i ) / r( j , i )
                dy = -1.0d0 * dr( j , i ) * ( zo( j , i ) -
     &                              zo( ntop , i ) ) /
     &                r( j , i )

                ro( j , i ) = ro( j , i ) + smooth * dx
                zo( j , i ) = zo( j , i ) + smooth * dy

 130     continue

        endif

        nnodes = nleft


c  Compute the output coordinate filenames
c  ---------------------------------------

        open ( 1 , file = 'crdfls.dt' )
        write( 1 , * ) nnodes
        write( 1 , * ) numfls
        write( 1 , * ) tmstep
        write( 1 , * ) pincep
        write( 1 , * ) nfltyp
```

```
      write( 1 , * ) phiacc
      write( 1 , * ) tinit
      write( 1 , * ) pvap
      write( 1 , * ) densty
      write( 1 , * ) ndatyp
      write( 1 , * ) gamma

      do 160 i = 1 , numfls , 1

          write( fmt = 2000 , unit = crdflo( i ) ) i
          write( 1 , 500 ) crdflo( i )

160   continue
      close( 1 )

c  Write the output coordinates to files
c  -------------------------------------

      do 170 i = 1 , numfls , 1

          open( 2 , file = crdflo( i ) )
          write( * , * ) 'Output coordinate file: ' , crdflo( i )

          do 170 j = nright , nleft , 1

          write( 2 , 530 ) ro( j , i ) * pixel ,
     &                     zo( j , i ) * pixel

170   continue
      close( 2 )

c  Housekeeping
c  ------------

500   format( a11 )
510   format( / )
520   format( i4 , t10 , i4 , t18 , i4 )
530   format( 1x , 2f14.7 )
540   format( t17 , i5 , t31 , i5 , t98 , i4 )
550   format( 5f14.7 )
2000  format( i8 )

      stop
      end
```

Appendix C

PRESSURE EXTRACTION PROGRAM

```
c     ****************************************************************
c     *                                                              *
c     *             MECH 955 - MAJOR ME THESIS  (1993)               *
c     *                 Course code 303 / ME(Hons)                   *
c     *                                                              *
c     *                  STEVEN HARVEY    8421248                    *
c     *                                                              *
c     *                 Supervised by Dr W. K. Soh                   *
c     *                                                              *
c     ****************************************************************
c     *                         Program BUBB11                       *
c     ****************************************************************
c     *                                                              *
c     * This program computes the surface pressure of a series of    *
c     * underwater vapour cavity images captured using high-speed    *
c     * cinematography.                                              *
c     *                                                              *
c     * The boundary integral method (bim) is utilised in an         *
c     * inverse way whereby the velocity potentials are computed     *
c     * using only bubble geometry and surface propagation data.     *
c     *                                                              *
c     * Subroutines supplied by J Best (MRL) and A Kucera (ADFA)     *
c     * are linked at compile time.                                  *
c     *                                                              *
c     ****************************************************************
```

```
c       Variable list
c       -------------

c       crdfl()  = coordinate filename array                character
c       densty   = fluid density                            dbl prec
c       distpf(0:,) = norm. dist. from (i+1)th to ith bub.  dbl prec
c       distpb(0:,) = norm. dist. from (i-1)th to ith bub.  dbl prec
c       dn       = used in subroutine PHI                   dbl prec
c       dincep   = distance from centroid to rigid b'dry    dbl prec
c       displ    = distance between node and n.i.p.         dbl prec
c       dpdnth(0:,) = theoretical normal velocity array     dbl prec
c       dphidn(0:,) = normal surface velocity array         dbl prec
c       dphidt(0:,) = time rate of change of vel. pot.      dbl prec
c       dphidz(0:,) = tangential surface velocity           dbl prec
c       dscale   = density scale                            dbl prec
c       dtscal   = time rate of change of vel. pot. scale   dbl prec
c       dummyd() = dummy array                              dbl prec
c       dxscal   = tangential velocity scale                dbl prec
c       energk() = kinetic energy of the bubble             dbl prec
c       errdpa   = max. abs. error in computed dphi/dn      dbl prec
c       errdpr   = max. rel. error in computed dphi/dn      dbl prec
c       errpha   = max. abs. error in computed phi          dbl prec
c       errphr   = max. rel. error in computed phi          dbl prec
c       errpra   = max. abs. error in computed pressure     dbl prec
c       errprr   = max. rel. error in computed pressure     dbl prec
c       escale   = energy scale                             dbl prec
c       fn       = solution function value                  dbl prec
c       gamma    = ratio of specific heats                  dbl prec
c       i        = file loop counter                        integer
c       j        = node loop counter                        integer
c       js       = solution node dummy                      integer
c       jn       = solution node dummy                      integer
c       jsolb(0:,)  = node ahead of i.p. - bwd              integer
c       jsolf(0:,)  = node ahead of i.p. - fwd              integer
c       maxfil   = no. coord. files dimensioning variable   integer
c       maxnod   = no. nodes dimensioning variable          integer
c       nbr      = if root not found=1, bwd displ.          integer
c       nberr    = no. bwd traj. errors                     integer
c       ndatyp   = data type, 1-exp., 0-validification      integer
c       nderr    = no. spurious dphi/dn errors              integer
c       ndphdt   = no. spurious Dphi/Dt errors              integer
c       nferr    = no. fwd traj. errors                     integer
c       nfr      = if root not found=1, fwd trajectory      integer
c       ni       = segment length parameter                 integer
```

```
c       nj        = arclength integration parameter          integer
c       nnodes    = total no. of surface nodes - 1           integer
c       nperr     = no. of spurious vel. pot. values         integer
c       npf       = if root not found=1, fwd displ.          integer
c       npferr    = no. project-fwd displ. errors            integer
c       npres     = no. spurious pressure errors             integer
c       nsurf     = 1-linear, 2-quadratic, 3-cubic elements  integer
c       numfls    = no. of coord. files to be read in        integer
c       pa(0:)    = cubic spline parameter array             dbl prec
c       pav()     = mean nodal pressure array                dbl prec
c       pb(0:)    = cubic spline parameter array             dbl prec
c       pc(0:)    = cubic spline parameter array             dbl prec
c       pd(0:)    = cubic spline parameter array             dbl prec
c       phiacc    = test condition for spurious errors       dbl prec
c       phic(0:,)   = velocity potential array               dbl prec
c       phitb(0:,)  = actual velocity pot. at new pos.- bwd  dbl prec
c       phitf(0:,)  = actual velocity pot. at new pos.- fwd  dbl prec
c       phith(0:,)  = theoretical velocity potential array   dbl prec
c       phix      = velocity potential dummy                 dbl prec
c       pi        = trigonometric pi                         dbl prec
c       pincep    = hydrostatic fluid pressure at inception  dbl prec
c       press(0:,)  = nodal pressure array                   dbl prec
c       presth()  = theoretical pressure array               dbl prec
c       pscale    = pressure scale                           dbl prec
c       psph()    = spherical bubble pressure array          dbl prec
c       pvap      = vapour pressure                          dbl prec
c       pwe()     = pressure array using work method         dbl prec
c       ra(0:,)   = cubic spline parameter array             dbl prec
c       rad(0:,)  = nodal radius array                       dbl prec
c       rav()     = mean radius array                        dbl prec
c       rb(0:,)   = cubic spline parameter array             dbl prec
c       rc(0:,)   = cubic spline parameter array             dbl prec
c       rd(0:,)   = cubic spline parameter array             dbl prec
c       rddot()   = mean surface radial acceleration array   dbl prec
c       rdot()    = mean surface radial velocity array       dbl prec
c       rdscal    = mean surface radial acceleration scale   dbl prec
c       rf        = node of interest radial coord.           dbl prec
c       ri(0:,)   = radial coordinate array                  dbl prec
c       rinit     = initial bubble radius                    dbl prec
c       rip1      = (i+1)th node radial coord.               dbl prec
c       rmaxi     = maximum bubble radius                    dbl prec
c       rn        = normal intersection point dummy          dbl prec
c       rnipb(0:,)  = normal intersection point - bwd        dbl prec
c       rnipf(0:,)  = normal intersection point - fwd        dbl prec
```

```
c       s(0:,)    = segment arclength array                          dbl prec
c       sgn       = direction indicator - +ve if expanding           dbl prec
c       sn        = solution segment arclength dummy                 dbl prec
c       solf      = fwd traj. arclength solution dummy               dbl prec
c       ssolb(0:,)  = bwd i.p. arclength soln. array                 dbl prec
c       ssolf(0:,)  = fwd i.p. arclength soln. array                 dbl prec
c       sum       = average pressure dummy variable                  dbl prec
c       t(0:,)    = arclength parameter array                        dbl prec
c       tangr     = radial vector                                    dbl prec
c       tangz     = vertical vector                                  dbl prec
c       theta(0:,)  = linear nodal direction array                   dbl prec
c       time()    = time array                                       dbl prec
c       tmstep    = timestep between subsequent coord. files         dbl prec
c       tr        = tangential component                             dbl prec
c       tscale    = time scale                                       dbl prec
c       tsolb(0:,)  = bwd i.p. arclength solution array              dbl prec
c       tsolf(0:,)  = fwd i.p. arclength solution array              dbl prec
c       tz        = tangential component                             dbl prec
c       v0        = initial volume ( theoretical )                   dbl prec
c       vnscal    = normal velocity scale                            dbl prec
c       vol       = dummy volume variable                            dbl prec
c       volume()  = computed volume array                            dbl prec
c       vpscal    = velocity potential scale                         dbl prec
c       vscale    = volume scale                                     dbl prec
c       xnr       = normal component                                 dbl prec
c       xnz       = normal component                                 dbl prec
c       xscale    = length scale                                     dbl prec
c       za(0:,)   = cubic spline parameter array                     dbl prec
c       zb(0:,)   = cubic spline parameter array                     dbl prec
c       zc(0:,)   = cubic spline parameter array                     dbl prec
c       zd(0:,)   = cubic spline parameter array                     dbl prec
c       zf        = node of interest vertical coordinate             dbl prec
c       zi(0:,)   = vertical coordinate array                        dbl prec
c       zip1      = (i+1)th node vertical coord.                     dbl prec
c       zn        = normal intersection point dummy                  dbl prec
c       znipb(0:,)  = normal intersection point - bwd                dbl prec
c       znipf(0:,)  = normal intersection point - fwd                dbl prec

c  last update 26/10/93  sbh
```

```
      program bubb11


c  Dimension variables
c  --------------------

      implicit double precision ( a - h , o - z )
      parameter( maxnod = 38 , maxfil = 200 )

      character*11 crdfl( maxfil )

      dimension ri( 0 : maxnod , maxfil ) ,
     &          zi( 0 : maxnod , maxfil ) ,
     &          dphidn( 0 : maxnod , maxfil ) ,
     &          s( 0 : maxnod , maxfil ) ,
     &          t( 0 : maxnod , maxfil ) ,
     &          phic( 0 : maxnod , maxfil ) ,
     &          ra( 0 : maxnod , maxfil ) ,
     &          rb( 0 : maxnod , maxfil ) ,
     &          rc( 0 : maxnod , maxfil ) ,
     &          rd( 0 : maxnod , maxfil ) ,
     &          za( 0 : maxnod , maxfil ) ,
     &          zb( 0 : maxnod , maxfil ) ,
     &          zc( 0 : maxnod , maxfil ) ,
     &          zd( 0 : maxnod , maxfil ) ,
     &          pa( 0 : maxnod , maxfil ) ,
     &          pb( 0 : maxnod , maxfil ) ,
     &          pc( 0 : maxnod , maxfil ) ,
     &          pd( 0 : maxnod , maxfil )
      dimension
     &          volume( maxfil ) ,
     &          dphidt( 0 : maxnod , maxfil ) ,
     &          press( 0 : maxnod , maxfil ) ,
     &          distpf( 0 : maxnod , maxfil ) ,
     &          distpb( 0 : maxnod , maxfil ) ,
     &          dphidz( 0 : maxnod , maxfil ) ,
     &          ssolf( 0 : maxnod , maxfil ) ,
     &          ssolb( 0 : maxnod , maxfil ) ,
     &          rnipf( 0 : maxnod , maxfil ) ,
     &          znipf( 0 : maxnod , maxfil ) ,
     &          rnipb( 0 : maxnod , maxfil ) ,
     &          znipb( 0 : maxnod , maxfil ) ,
     &          phitf( 0 : maxnod , maxfil ) ,
     &          phitb( 0 : maxnod , maxfil ) ,
```

```
     &            jsolf( 0 : maxnod , maxfil ) ,
     &            jsolb( 0 : maxnod , maxfil ) ,
     &            energk( maxfil ) ,
     &            tsolf( 0 : maxnod , maxfil ) ,
     &            tsolb( 0 : maxnod , maxfil ) ,
     &            dummyd( maxfil ) ,
     &            presth( maxfil ) ,
     &            dpdnth( 0 : maxnod , maxfil ) ,
     &            phith( 0 : maxnod , maxfil ) ,
     &            theta( 0 : maxnod , maxfil ) ,
     &            rav( maxfil ) ,
     &            rad( 0 : maxnod , maxfil ) ,
     &            pav( maxfil ) ,
     &            pwe( maxfil ) ,
     &            rdot( maxfil ) ,
     &            rddot( maxfil ) ,
     &            psph( maxfil ) ,
     &            time( maxfil )


c  Display title screen
c  --------------------

      write( * , * ) 'V A P O U R  C A V I T Y  P R E S S U R E'
      write( * , * )
      write( * , * ) '       C O M P U T A T I O N'
      write( * , * )
      write( * , * ) '          by  S. B. Harvey'
      write( * , * )
      write( * , * )
      write( * , * ) '         v1.03    26 October 1993'
      write( * , * )
      write( * , * )
      write( * , * ) 'Please enter  1 - linear representation'
      write( * , * ) '              2 - quadratic representation'
      write( * , * ) '              3 - cubic representation'
      write( * , * )
      read( * , * ) nsurf
      write( * , * )
```

```
c  Initialise variables
c  --------------------

       pi = 3.14159265358979 32d0
       ni = 4
       nj = 4
       dn = -1.0d0

       write( * , * ) 'now reading in coordinate files...'
       write( * , * )


c  Read in the coordinate data and scale them
c  ------------------------------------------

       call indata( ri , zi , numfls , crdfl , rad , nsurf ,
     &              tmstep , nnodes , maxfil , maxnod ,
     &              pincep , dpdnth , phith , phiacc ,
     &              pvap , rmaxi , densty , pscale , tscale ,
     &              escale , vscale , vnscal , xscale , dscale ,
     &              dincep , rinit  , v0 ,
     &              time , ndatyp , vpscal , dxscal , dtscal ,
     &              rdscal , gamma )


       if( nsurf .eq. 1 ) then

c        set up linear elements
c        ----------------------
         write( * , * ) 'now setting up linear elements...'
         write( * , * )

         do 10 i = 1 , numfls , 1

            t( 0 , i ) = 0.0d0

            do 2 j = 1 , nnodes , 1

               s( j  , i ) = sqrt( ( ri( j , i ) - ri( j - 1 , i ) )
     &                     ** 2 + ( zi( j , i ) - zi( j - 1 , i ) ) ** 2 )
               t( j , i ) = t( j - 1 , i ) + s( j , i )

2           continue
```

```
            call linear( ri( 0 , i )  , zi( 0 , i ) ,
     &                   s( 0 , i )  , t( 0 , i ) ,
     &                   maxnod ,
     &                   ra( 0 , i ) , rb( 0 , i ) ,
     &                   rc( 0 , i ) , rd( 0 , i ) ,
     &                   nnodes ,
     &                   za( 0 , i ) , zb( 0 , i ) ,
     &                   zc( 0 , i ) , zd( 0 , i ) ,
     &                   dn , theta( 0 , i ) )

 10       continue


c         compute the volumes of the bubbles using their linear
c         ------------------------------------------------------
c         representation
c         --------------
          do 11 i = 1 , numfls , 1

             call voluml( maxnod , nnodes ,
     &                    ra( 0 , i ) , za( 0 , i ) ,
     &                    vol )

             volume( i ) = vol
             presth( i )  = ( v0 / vol ) ** gamma

 11       continue

        elseif( nsurf .eq. 2 ) then

c         set up quadratic elements (not yet implemented)
c         -----------------------------------------------
          goto 15

        elseif( nsurf .eq. 3 ) then

c         set up cubic elements
c         ---------------------
 15       write( * , * ) 'now setting up cubic elements...'
          write( * , * )
```

```
  do 18 i = 1 , numfls , 1

            call arcl( maxnod , nnodes ,  nj , ni , dn ,
     &                 ra( 0 , i ) , rb( 0 , i ) , rc( 0 , i ) ,
     &                 rd( 0 , i ) , ri( 0 , i ) , s( 0 , i ) ,
     &                 za( 0 , i ) , zb( 0 , i ) , zc( 0 , i ) ,
     &                 zd( 0 , i ) , zi( 0 , i ) , t( 0 , i ) )

 18       continue

c         compute the volumes of the bubbles using their cubic
c         ----------------------------------------------------
c         representation
c         --------------
          do 19 i = 1 , numfls , 1

              call volumc( maxnod , nnodes , ra( 0 , i ) , rb( 0 , i )
     &                   , rc( 0 , i ) , rd( 0 , i ) , za( 0 , i )
     &                   , zb( 0 , i ) , zc( 0 , i ) , zd( 0 , i )
     &                   , s( 0 , i ), vol )

              volume( i ) = vol
              presth( i )  = ( v0 / vol ) ** gamma

 19       continue

        endif


c  Compute the mean radius of each bubble shape using its volume
c  -------------------------------------------------------------

        do 20 i = 1 , numfls , 1

            rav( i ) = ( 3.0d0 * volume( i ) / ( 4.0d0 * pi ) ) **
     &                 ( 1.0d0 / 3.0d0 )

 20     continue
```

```
c  Smooth the radii timewise
c  -------------------------

        do 21 i = 2 , numfls - 1 , 1

            dummyd( i ) = 0.5d0 * ( rav( i - 1 ) + rav( i + 1 ) )

 21     continue

        do 22 i = 2 , numfls - 1 , 1

            rav( i ) = dummyd( i )

 22     continue


c  Compute radial derivatives
c  --------------------------

       do 23 i = 2 , numfls - 1 , 1

           rdot( i ) = 1.0d0 / ( 2.0d0 * tmstep ) * ( rav( i + 1 ) -
     &                              rav( i - 1 ) )
           rddot( i ) = 1.0d0 / tmstep ** 2 * ( rav( i + 1 ) -
     &                             2.0d0 * rav( i ) + rav( i - 1 ) )

 23     continue


c  Smooth the rddot values timewise
c  --------------------------------

        do 24 i = 2 , numfls - 1 , 1

            dummyd( i ) = 0.5d0 * ( rddot( i - 1 ) + rddot ( i + 1 ) )

 24    continue

       do 25 i = 2 , numfls - 1 , 1

           rddot( i ) = dummyd( i )

 25    continue
```

```
c  Compute the distpf and distpb arrays
c  ----------------------------------

c  First project 'back' from the current bubble surface to the previous
c  one and compute the backward projected normal distance.
c  note that this is a signed distance.  It will be positive
c  if the bubble is contracting.

      write( * , * ) 'normal velocity calculation - backwards'

      do 30 i = 2 , numfls , 1

         write( * , * ) 'i = ' , i
         distpb( 0 , i ) = -1.0 * ( zi( 0 , i )
     &                        - zi( 0 , i - 1 ) )
         distpb( nnodes , i ) = zi( nnodes , i )
     &                        - zi( nnodes , i - 1 )

         rnipb( 0 , i )      = 0.0d0
         znipb( 0 , i )      = zi( 0 , i - 1 )
         rnipb( nnodes , i ) = 0.0d0
         znipb( nnodes , i ) = zi( nnodes , i - 1 )
         jsolb( 0 , i )      = 1
         jsolb( nnodes , i ) = nnodes + 1
         tsolb( 0 , i )      = t( 0 , i - 1 )
         tsolb( nnodes , i ) = t( nnodes , i - 1 )

         do 30 j = 1 , nnodes - 1 , 1

            rf = ri( j , i )
            zf = zi( j , i )

            call projbk( ra( 0 , i - 1 ) , rb( 0 , i - 1 ) ,
     &                   rc( 0 , i - 1 ) , rd( 0 , i - 1 ) ,
     &                   za( 0 , i - 1 ) , zb( 0 , i - 1 ) ,
     &                   zc( 0 , i - 1 ) , zd( 0 , i - 1 ) ,
     &                   ri( 0 , i - 1 ) , zi( 0 , i - 1 ) ,
     &                   s( 0 , i - 1 ) , rf , zf , maxnod ,
     &                   displ , sn , js , rn , zn , nnodes )

            ssolb( j , i ) = sn
            jsolb( j , i ) = js
            tsolb( j , i ) = t( jsolb( j , i ) - 1 , i - 1 )
     &                       + ssolb( j , i )
```

```
                rnipb( j , i ) = rn
                znipb( j , i ) = zn
                distpb( j , i ) = displ

 30     continue


c  Secondly project 'forward' from the current bubble surface to the
c  next one and compute the forward projected normal distance.  Note
c  that this distance is signed in each case (+ve if contracting).

        write( * , * ) 'normal velocity calculation - forwards'

        do 40 i = 1 , numfls - 1 , 1

         write( * , * ) 'i = ' , i

         distpf( 0 , i ) = -1.0 * ( zi( 0 , i + 1 ) - zi( 0 , i ))
         distpf( nnodes , i ) = zi( nnodes , i + 1 )
     &                        - zi( nnodes , i )

         rnipf( 0 , i )      = 0.0d0
         znipf( 0 , i )      = zi( 0 , i + 1 )
         rnipf( nnodes , i ) = 0.0d0
         znipf( nnodes , i ) = zi( nnodes , i + 1 )
         jsolf( 0 , i )      = 1
         jsolf( nnodes , i ) = nnodes + 1
         tsolf( 0 , i )      = t( 0 , i + 1 )
         tsolf( nnodes , i ) = t( nnodes , i + 1 )

             do 40 j = 1 , nnodes - 1 , 1

                 rf = ri( j , i )
                 zf = zi( j , i )
                 tr = rb( j , i )
                 tz = zb( j , i )
                 rip1 = ri( j , i + 1 )
                 zip1 = zi( j , i + 1 )
```

```
            call projfw( ra( 0 , i + 1 ) , rb( 0 , i + 1 ) ,
     &                   rc( 0 , i + 1 ) , rd( 0 , i + 1 ) ,
     &                   za( 0 , i + 1 ) , zb( 0 , i + 1 ) ,
     &                   zc( 0 , i + 1 ) , zd( 0 , i + 1 ) ,
     &                   s( 0 , i + 1 ) , rf , zf ,
     &                   tr , tz , maxnod , displ , nnodes ,
     &                   sn , js , j , rip1 , zip1 , npf )

      npferr = npferr + npf

      ssolf( j , i ) = sn
      jsolf( j , i ) = js
      tsolf( j , i ) = t( jsolf( j , i ) - 1 , i + 1 ) + ssolf( j , i )

      distpf( j , i ) = displ

      if( npf .eq. 1 ) then
          write( * , * ) 'root not found'
      endif

 40   continue


c  Compute dphi/dn values
c  ----------------------

      do 50 i = 1 , numfls , 1

          do 60 j = 0 , nnodes , 1

              dphidn( j , i ) = -1.0 * ( ( distpf( j , i ) +
     &                                     distpb( j , i ) ) /
     &                                   ( 2.0 * tmstep ) )

 60       continue

 50   continue
```

```
c  Filter out any spurious dphi/dn values and write dphi/dn to file
c  ------------------------------------------------------------------

       open( 1 , file = 'norm_vel.dat' )
       write( 1 , * ) 'time      j       dphidn'
       nderr = 0

       do 70 i = 2 , numfls - 1 , 1

       write( 1 , 1110 ) time( i ) * tscale , 0 ,
     &      dphidn( 0 , i ) * vnscal

       do 68 j = 1 , nnodes - 1 , 1

           if( abs( dphidn( j , i ) ) .gt. 5.0d0 *
     &         abs( dphidn( 0 , i ) ) ) then
               sgn = ( volume( i ) - volume( i + 1 ) ) /
     &               abs( volume( i ) - volume( i + 1 ) )
               dphidn( j , i ) = sgn * dist( ri( j , i - 1 ) ,
     &                                       zi( j , i - 1 ) ,
     &                                       ri( j , i + 1 ) ,
     &                                       zi( j , i + 1 ) ) /
     &                                       ( 2.0d0 * tmstep )
               nderr = nderr + 1
           endif

           if( abs( dphidn( j , i ) ) .eq. 0.0d0 ) then
               sgn = ( volume( i ) - volume( i + 1 ) ) /
     &                 abs( volume( i ) - volume( i + 1 ) )
               dphidn( j , i ) = sgn * dist( ri( j , i - 1 ) ,
     &                                       zi( j , i - 1 ) ,
     &                                       ri( j , i + 1 ) ,
     &                                       zi( j , i + 1 ) ) /
     &                                       ( 2.0d0 * tmstep )
               nderr = nderr + 1
           endif

           write( 1 , 1110 ) time( i ) * tscale , j ,
     &                       dphidn( j , i ) * vnscal

68     continue
```

```
      write( 1 , 1110 ) time( i ) * tscale , nnodes ,
     &      dphidn( nnodes , i ) * vnscal

 70   continue

      close( 1 )

      write( * , * )
      write( * , * ) 'no. projfw errors = ' , npferr
      write( * , * ) 'no. dphi/dn spurious errors = ' , nderr
      write( * , * ) 'dphi/dn values computed'


c  Renumber the nodes 1 to nnodes+1 and call subroutine PHI (see Best,
c  ------------------------------------------------------------------
c  1991b for a complete source listing).
c  ------------------------------------
c  Velocity potentials (phic) are returned.
c  ---------------------------------------

        do 90 i = 1 , numfls , 1

           write( * , * ) 'computing phi for: ' , time( i )

          call phi( ri( 0 , i ) , zi( 0 , i ) ,
     &              dphidn( 0 , i ) , phic( 0 , i ) ,
     &              nnodes + 1 , nnodes )

 90    continue


c  Write phic to file
c  ------------------

      open( 1 , file = 'phi.dat' )
      write( 1 , * ) 'time  j       phi'

      do 91 i = 1 , numfls , 1

       do 91 j = 0 , nnodes , 1

          write( 1 , 1110 ) time( i ) * tscale , j ,
     &                      phic( j , i ) * vpscal
```

```
 91    continue

       close( 1 )

c  Compute the kinetic energy of the bubble using linear elements
c  ------------------------------------------------------------

       write( * , * ) 'computing kinetic energy'

       do 100 i = 1 , numfls , 1

           do 105 j = 1 , nnodes - 1 , 1

               dummyd( j ) = -1.0d0 * dphidn( j , i )
               pa( j , i ) = phic( j , i )
               pb( j , i ) = ( phic( j + 1 , i )  - phic( j , i ) ) /
     &                       s( j + 1 , i )
               pc( j , i ) = 0.0d0
               pd( j , i ) = 0.0d0

 105       continue

           dummyd( nnodes ) = -1.0d0 * dphidn( nnodes , i )

           call energ( ra( 0 , i ) , rb( 0 , i ) , rc( 0 , i ) ,
     &                 rd( 0 , i ) ,
     &                 pa( 0 , i ) , pb( 0 , i ) ,
     &                 pc( 0 , i ) , pd( 0 , i ) , dummyd ,
     &                 energk( i ) ,
     &                 t( 0 , i ) , maxnod , nnodes )

 100   continue
```

```
c  Compute the tangential velocity
c  -----------------------------

       call dphdz( dphidz , t , phic ,
     &             numfls , maxfil , nnodes , maxnod )

c  Check quality of data before computing trajectories
c  -------------------------------------------------

       if( nderr .gt. int( numfls / 2 ) ) then

        write( * , * )
        write( * , * ) 'too many dphi/dn errors - estimating dphi/dt'

          do 106 i = 3 , numfls - 2 , 1

            do 106 j = 0 , nnodes , 1

                 phitf( j , i ) = ( phic( j , i + 1 ) )
                 phitb( j , i ) = ( phic( j , i - 1 ) )

 106      continue

        goto 125

       endif


c  Compute the actual trajectory of a fluid particle travelling
c  -----------------------------------------------------------
c                      -              -              -
c  forward at velocity v = (dphi/dn) n + (dphi/dxi) t
c  -------------------------------------------------

      do 112 i = 3 , numfls - 2 , 1

          write( * , * ) 'forward trajectory: ' , time( i )

          phitf( 0 , i ) = phic( 0 , i + 1 )
          phitf( nnodes , i ) = phic( nnodes , i + 1 )
```

```
      do 110 j = 1 , nnodes - 1 , 1

          rf = ri( j , i )
          zf = zi( j , i )
          xnr = -1.0d0 * zb( j , i )
          xnz = rb( j , i )
          tr = rb( j , i )
          tz = zb( j , i )

          tangr = -1.0d0 * ( dphidn( j , i ) * xnz +
     &                       dphidz( j , i ) * tz )

          tangz =          ( dphidn( j , i ) * xnr +
     &                       dphidz( j , i ) * tr )

          call fortrj( ra( 0 , i + 1 ) , rb( 0 , i + 1 ) ,
     &                 rc( 0 , i + 1 ) , rd( 0 , i + 1 ) ,
     &                 za( 0 , i + 1 ) , zb( 0 , i + 1 ) ,
     &                 zc( 0 , i + 1 ) , zd( 0 , i + 1 ) ,
     &                 s( 0 , i + 1 ) , rf , zf ,
     &                 phic( 0 , i + 1 ) ,
     &                 tangr , tangz , maxnod , phix , j ,
     &                 sn , fn , jn , nfr , nnodes )

          phitf( j , i ) = phix
          nferr = nferr + nfr


c  If a satisfactory value for the forward trajectory phi is not found
c  ------------------------------------------------------------------

          if ( ( ( ( abs( phitf( j , i ) - phic( j , i + 1 ) ) )
     &           / abs( phic( j , i + 1 ) ) ) .gt. phiacc ) .or.
     &           ( nfr .eq. 1 ) ) then
              write( * , * ) 'recalculating fwd trajectory i = ' , i
              write( * , * ) '                              j = ' , j
              write( * , * )

c             if dphidz is negative then the t.i.p. must lie
c             between the n.i.p. and jsol

              if ( dphidz( j , i ) .lt. 0.0d0 ) then
               solf = ssolf( j , i ) + dphidz( j , i ) * tmstep
              else
```

```
              solf = ssolf( j , i ) - dphidz( j , i ) * tmstep
             endif

              phitf( j , i ) = pa( jsolf( j , i ) - 1 , i + 1 ) +
     &                         pb( jsolf( j , i ) - 1 , i + 1 ) *
     &                         solf

            endif

 110       continue

 112    continue


c  Compute the actual trajectory of a fluid particle travelling
c  ------------------------------------------------------------
c                                 -                -                -
c  backward at velocity v = (dphi/dn) n + (dphi/dxi) t
c  ------------------------------------------------


      do 120 i = 3 , numfls - 2 , 1

          write( * , * ) 'backward trajectory: ' , time( i )

          phitb( 0 , i ) = phic( 0 , i - 1 )
          phitb( nnodes , i ) = phic( nnodes , i - 1 )

          do 115 j = 1 , nnodes - 1 , 1

              rf = ri( j , i )
              zf = zi( j , i )

              call baktrj( ra( 0 , i - 1 ) , rb( 0 , i - 1 ) ,
     &                     rc( 0 , i - 1 ) , rd( 0 , i - 1 ) ,
     &                     za( 0 , i - 1 ) , zb( 0 , i - 1 ) ,
     &                     zc( 0 , i - 1 ) , zd( 0 , i - 1 ) ,
     &                     ri( 0 , i - 1 ) , zi( 0 , i - 1 ) ,
     &                     dphidn( 0 , i ) ,
     &                     phic( 0 , i - 1 ) ,
     &                     dphidz( 0 , i ) ,
     &                     s( 0 , i - 1 ) , rf , zf ,
     &                     maxnod , nnodes , phix , nbr )
```

```
            phitb( j , i ) = phix
            nberr = nberr + nbr

 115        continue

 120    continue


c  Compute the time rate of change of the velocity potential Dphi/Dt
c  ------------------------------------------------------------------

 125    call dphdt( dphidt , phitf , phitb , tmstep , ndphdt ,
     &              numfls , maxfil , nnodes , maxnod , phiacc )


c  Compute the nodal pressures
c  ---------------------------

        npres = 0

        do 126 i = 3 , numfls - 2 , 1

        do 127 j = 0 , nnodes , 1

        press( j , i ) =( 1.0d0 +  5.0d-1 * ( dphidn( j , i )
     &                      ** 2.0d0 + dphidz( j , i )
     &                      ** 2.0d0 ) - dphidt( j , i ) )

        if( abs( press( j , i ) ) .gt. 10.0d0 *
     &             abs( press( 0 , i ) ) ) then
           press( j , i ) = press( 0 , i )
           npres = npres + 1
        endif

 127        continue

 126    continue
```

```
      do 129 i = 3 , numfls - 2 , 1

          sum = 0.0d0

          do 128 j = 0 , nnodes , 1
             sum = sum + press( j , i )
128       continue
      pav( i ) = sum / ( nnodes + 1 )

129   continue


      if( ndatyp .eq. 0 ) then

c      compute the absolute and relative errors in dphidn,
c      ---------------------------------------------------
c      phi and pressure if validation data is used
c      ------------------------------------------
       open( 5 , file = 'dpdn_a.dat' )
       open( 6 , file = 'dpdn_r.dat' )
       open( 7 , file = 'phi_a.dat' )
       open( 8 , file = 'phi_r.dat' )
       open( 9 , file = 'press_a.dat' )
       open( 10 , file = 'press_r.dat' )

       write( 5 , * ) 'maximum absolute error in dphidn'
       write( 6 , * ) 'maximum relative error in dphidn'
       write( 7 , * ) 'maximum absolute error in phi'
       write( 8 , * ) 'maximum relative error in phi'
       write( 9 , * ) 'maximum absolute error in pressure'
       write( 10 , * ) 'maximum relative error in pressure'

       do 132 i = 1 , numfls , 1

           errdpa = 0.0d0
           errdpr = 0.0d0
           errpha = 0.0d0
           errphr = 0.0d0
           errpra = 0.0d0
           errprr = 0.0d0
```

```
          do 133 j = 0 , nnodes , 1

             if( dpdnth( j , i ) .ne. 0.0d0 ) then

              errdpa = max( errdpa , abs( dpdnth( j , i ) +
     &         dphidn( j , i ) ) )

              errdpr = max( errdpr ,  abs( dpdnth( j , i ) +
     &         dphidn( j , i ) ) / abs( dpdnth( j , i ) ) )

             endif

             if( phith( j , i ) .ne. 0.0d0 ) then

              errpha = max( errpha , abs( phith( j , i ) -
     &         phic( j , i ) ) )

              errphr = max( errphr , abs( phith( j , i ) -
     &         phic( j , i ) ) / phith( j , i ) )

             endif

             if( presth( i ) .ne. 0.0d0 ) then

              errpra = max( errpra , abs( presth( i ) -
     &         press( j , i ) ) )

              errprr = max( errprr , abs( presth( i ) -
     &         press( j , i ) ) / presth( i ) )

             endif

133       continue

          write( 5 , 1020 ) time( i ) , errpa
          write( 6 , 1020 ) time( i ) , errdpr
          write( 7 , 1020 ) time( i ) , errpha
          write( 8 , 1020 ) time( i ) , errphr
          write( 9 , 1020 ) time( i ) , errpra
          write( 10 , 1020 ) time( i ) , errprr

132     continue
```

```
      close( 5 )
      close( 6 )
      close( 7 )
      close( 8 )
      close( 9 )
      close( 10 )

      endif

      write( * , * )
      write( * , * ) 'pressures computed'
      write( * , * ) 'forward trajectory errors = ' , nferr
      write( * , * ) 'backward trajectory errors = ' , nberr
      write( * , * ) 'dphidt errors = ' , ndphdt
      write( * , * ) 'pressure errors = ' , npres

      write( * , * ) 'writing data to files...'

c  Use the spherical bubble approximation and work method
c  ------------------------------------------------------
c  to compute average pressure
c  ---------------------------

      p1 = pav( 3 )

      do 135 i = 3 , numfls - 2 , 1

         pwe( i + 1 ) = 2.0d0 * ( energk( i + 1 ) - energk( i ) ) /
     &                        ( volume( i + 1 ) - volume ( i ) ) -
     &                          p1
         p1 = pwe( i + 1 )
         psph( i ) = ( rav( i ) * rddot( i ) + 1.5d0 *
     &                 rdot( i ) ** 2 + 1.0d0 /
     &                 ( 2.0d0 * abs( dincep ) )
     &                 * rav( i ) * ( rav( i )
     &                 * rddot( i ) +
     &                 2.0d0 * rdot( i ) ** 2 ) + 1.0d0 )

 135  continue
```

```
c  Write results to files
c  ---------------------

      if( ndatyp .eq. 0 ) then
       open( 1 , file = 'norm_vel.dat' )
       open( 3 , file = 'phi.dat' )
       write( 1 , * ) 'time  j        dphidn     dpdnth err_a  err_r'
       write( 3 , * ) 'time  j        phi        phith  err_a  err_r'
      endif

      open( 2 , file = 'envol9.dat' )
      open( 4 , file = 'pressure.dat' )
      open( 5 , file = 'dphidn9.dat' )
      open( 6 , file = 'rav.dat' )
      open( 7 , file = 'dphidt.dat' )
      open( 9 , file = 'presth.dat' )

      write( 2 , * ) 'time     vol(m^3)     energy(j)  '
      write( 4 , * ) 'time  j        press    pressth err_a    err_r'
      write( 5 , * ) 'time   dphidn_9     dphids_9     phi_9'
      write( 6 , * ) 'time   dphidt_9     rav        rdot     rddot'
      write( 7 , * ) 'time   j      phitb    phic    phitf   dphidt'
      write( 9 , * ) 'time  pav(pa) pth(pa) pwe(pa) psph(pa)  r  a '

      do 140 i = 1 , numfls , 1

         write( 2 , 1030 ) time( i ) * tscale ,
     &                     volume( i ) * vscale ,
     &                     energk( i ) * escale
         write( 5 , 1050 ) time( i ) * tscale ,
     &                     dphidn( 9 , i ) * vnscal ,
     &                     dphidz( 9 , i ) * dxscal ,
     &                     phic( 9 , i ) * vpscal
         write( 6 , 1070 ) time( i ) * tscale ,
     &                     dphidt( 9 , i ) * dtscal ,
     &                     rav( i ) * xscale ,
     &                     rdot( i ) * vnscal ,
     &                     rddot ( i ) * rdscal

       if( ndatyp .eq. 1 ) then
```

```
c           experimental data used
c           ----------------------
            write( 9 , 1130 ) time( i ) * tscale ,
     &                        pav( i ) * pscale ,
     &                        presth( i ) * pscale ,
     &                        pwe( i ) * pscale ,
     &                        psph( i ) * pscale

          else

c           validation data used
c           --------------------
            write( 9 , 1080 ) time( i ) ,
     &                        pav( i ) ,
     &                        presth( i ) * pscale ,
     &                        pwe( i ) ,
     &                        psph( i ) ,
     &                      ( psph( i ) - presth( i ) * pscale ) /
     &                      ( presth( i ) * pscale ) ,
     &                        psph( i ) - presth( i ) * pscale

          endif

           do 140 j = 0 , nnodes , 1

              if( ndatyp .eq. 0 ) then

                   write( 1 , 1100 ) time( i ) , j ,
     &                               dphidn( j , i ) ,
     &                               -1.0d0 * dpdnth( j , i ) ,
     &                               abs( dpdnth( j , i ) +
     &                               dphidn( j , i ) ) ,
     &                               abs( dpdnth( j , i ) +
     &                         dphidn( j , i ) ) / dpdnth( j , i )

                   write( 3 , 1100 ) time( i ) , j ,
     &                               phic( j , i ) ,
     &                               phith( j , i ) ,
     &                               abs( phith( j , i ) -
     &                               phic( j , i ) ) ,
     &                               abs( phith( j , i ) -
     &                         phic( j , i ) ) / phith( j , i )
```

```
                write( 4 , 1100 ) time( i ) , j ,
     &                      press( j , i ) ,
     &                      presth( i ) * pscale ,
     &                      abs( presth( i ) * pscale -
     &                      press( j , i ) ) ,
     &                      abs( presth( i ) * pscale -
     &                  press( j , i ) ) / ( presth( i )*pscale )

           endif

           if( ndatyp .eq. 1 ) then

            write( 4 , 1120 ) time( i ) * tscale , j ,
     &                      press( j , i ) * pscale ,
     &                      presth( i ) * pscale

           endif

           write( 7 , 1100 ) time( i ) * tscale ,
     &                      j , phitb( j , i ) * vpscal ,
     &                          phic( j , i ) * vpscal ,
     &                          phitf( j , i ) * vpscal ,
     &                          dphidt( j , i ) * dtscal

140          continue

           if( ndatyp .eq. 0 ) then
               close( 1 )
               close( 3 )
           endif

           close( 2 )
           close( 4 )
           close( 5 )
           close( 6 )
           close( 7 )
           close( 9 )
```

```
c  Housekeeping
c  ------------

 1020   format( 2f14.7 )
 1030   format( 3f14.7 )
 1040   format( i3 , 5f14.7 )
 1050   format( 4f14.7 )
 1060   format( i3 , 4f14.7 )
 1070   format( f10.4 , 4f20.5 )
 1080   format( 7f10.5 )
 1090   format( f14.7 , i3 , 3f14.7 )
 1100   format( f14.7 , i3 , 4f14.7 )
 1110   format( f14.7 , i3 , f14.7 )
 1120   format( f14.3 , i5 , 2f20.4 )
 1130   format( f10.5 , 4f14.3 )

        stop
        end
```

```
c       ************************************************************
c       *                      Subroutine INDATA                   *
c       ************************************************************
c       *                                                          *
c       * INDATA firstly reads in the list of data filenames and   *
c       * then each individual coordinate file in sequence.  The   *
c       * coordinates must be in the format 1x,2f14.7, double      *
c       * precision, 19 nodes with node zero at the bottom.  The   *
c       * rigid boundary is at z=0. max no. of files = maxfil.     *
c       *                                                          *
c       * The coordinates are scaled to maximum radius.            *
c       *                                                          *
c       ************************************************************

c       Local variable list
c       -------------------

c       dum      = dummy variable                          dbl prec
c       ddum     = dummy variable                          dbl prec
c       idum     = dummy variable                          integer
c       nfltyp   = input file type, 0-long, 1-induv.       integer
c       rexp     = initial mean radius                     dbl prec
c       rsum     = sum of radii                            dbl prec
c       tinit    = time of first photographic frame        dbl prec

c last update 13/7/93  sbh


        subroutine indata( ri , zi , numfls , crdfl , rad , nsurf ,
     &                     tmstep , nnodes , maxfil , maxnod ,
     &                     pincep , dpdnth , phith , phiacc ,
     &                     pvap , rmaxi , densty , pscale , tscale ,
     &                     escale , vscale , vnscal , xscale , dscale ,
     &                     dincep , rinit  , vinit ,
     &                     time , ndatyp , vpscal , dxscal , dtscal ,
     &                     rdscal , gamma )
```

```
c  Dimension variables
c  -------------------

       implicit double precision ( a - h , o - z )

       character*11 crdfl( maxfil )

       dimension ri( 0 : maxnod , maxfil ) ,
     &           zi( 0 : maxnod , maxfil ) ,
     &           dpdnth( 0 : maxnod , maxfil ) ,
     &           phith( 0 : maxnod , maxfil ) ,
     &           rad( 0 : maxnod , maxfil ) ,
     &           time( maxfil )


c  Set constants
c  -------------

       dum = 0.0d0
       pi = 3.14159265358979 32d0
       rmaxi = 0.0d0


c  Open data files and read in coordinate data
c  -------------------------------------------

       open( 1 , file = 'crdfls.dt' )
       read( 1 , * ) nnodes
       read( 1 , * ) numfls
       read( 1 , * ) tmstep
       read( 1 , * ) pincep
       read( 1 , * ) nfltyp
       read( 1 , * ) phiacc
       read( 1 , * ) tinit
       read( 1 , * ) pvap
       read( 1 , * ) densty
       read( 1 , * ) ndatyp
       read( 1 , * ) gamma
```

```
c  If data is computed (for verification purposes) then ndatyp = 0
c  ------------------------------------------------------------

       if( ndatyp .eq. 0 ) then

        if ( nfltyp .ne. 1 ) then

c         if input file is long (nfltyp = 0) then read all (validation)
c         ------------------------------------------------------------
c         data in from the one file
c         -------------------------
            write( * , * ) 'reading in long file...'

            do 10 i = 1 , numfls , 1

                dum = dum + tmstep
                idum = int( 10000.0 * dum )
                write( fmt = 2000 , unit = crdfl( i ) ) idum

                write( * , 500 ) crdfl( i )
                read( 1 , * )  ddum

                do 11 j = 0 , nnodes , 1

                    read( 1 , 510 ) ri( j , i ) ,
     &                              zi( j , i ) ,
     &                              dpdnth( j , i ) ,
     &                              phith( j , i )

 11             continue

 10          continue

             close( 1 )
             goto 1000

        else
```

```
c         read in the filenames of the input files
c         ----------------------------------------
          do 20 i = 1 , numfls , 1

             read( 1 , 500 ) crdfl( i )

 20       continue

          endif

          close( 1 )

c         read in coordinates
c         -------------------
          do 30 i = 1 , numfls , 1

             open( 2 , file = crdfl( i ) )
             write( * , * ) 'coordinate file: ' , crdfl( i )

             do 35 j = 0 , nnodes , 1

                 read( 2 , 510 ) ri( j , i ) ,
     &                           zi( j , i ) ,
     &                           dpdnth( j , i ) ,
     &                           phith( j , i )

 35          continue

             close( 2 )

 30       continue

 1000     continue

          else

c         data is experimental
c         --------------------
            do 40 i = 1 , numfls , 1

                 read( 1 , 500 ) crdfl( i )

 40          continue
          close( 1 )
```

```
c         read in coordinates
c         -------------------
          do 50 i = 1 , numfls , 1

             open( 2 , file = crdfl( i ) )
             write( * , * ) 'coordinate file: ' , crdfl( i )

             do 45 j = 0 , nnodes , 1

                 read( 2 , 520 ) ri( j , i ) ,
     &                           zi( j , i )

 45          continue

             close( 2 )

 50       continue

          endif

          dincep = zi( nnodes / 2 , 1 )


c  Find the maximum radius
c  -----------------------

          do 55 i = 1 , numfls , 1

             do 55 j = 0 , nnodes , 1

                 rad( j , i ) = ( ri( j , i ) ** 2 + ( zi( j , i ) -
     &                          zi( nnodes/2 , i ) ) ** 2 ) ** 0.5

                 if ( rad( j , i ) .gt. rmaxi ) then
                     rmaxi = rad( j , i )
                 endif

 55       continue
```

```
c  Compute the initial mean radius
c  -----------------------------

        do 60 j = 0 , nnodes , 1

           rsum = rsum + rad( j , 1 )

 60     continue

        rexp = rsum / ( nnodes + 1 )


c  Compute other scale factors
c  --------------------------

        pscale = pincep - pvap
        escale = 0.5d0 * rmaxi ** 3 * pscale
        vnscal = ( pscale / densty ) ** 0.5
        vscale = rmaxi ** 3
        xscale = rmaxi
        dscale = 1.0d0
        vpscal = vnscal * xscale
        dxscal = vnscal
        dtscal = vnscal ** 2
        rdscal = pscale / ( densty * rmaxi )


c  Scale the coordinates to the maximum radius
c  -------------------------------------------

        do 75 i = 1 , numfls , 1

            do 75 j = 0 , nnodes , 1

                ri( j , i ) = ri( j , i ) / xscale
                zi( j , i ) = zi( j , i ) / xscale

 75     continue
```

```
c  Nondimensionalise some constants
c  ---------------------------------

      open( 2 , file = 'scale.dat' )
      write( * , * )
      write( * , * )

      if ( ndatyp . eq . 1 ) then
          write( * , * ) 'experimental data'
          write( 2 , * ) 'experimental data'
          write( * , * )
          tscale = rmaxi * ( densty / pscale ) ** 0.5
      else
          write( * , * ) 'verification data'
          write( 2 , * ) 'verification data'
          write( * , * )
          tscale = 1.0d0
          xscale = 1.0d0
      endif

      if( nsurf .eq. 1 ) then
          write( * , * ) 'linear representation'
          write( 2 , * ) 'linear representation'
          write( * , * )
      elseif(nsurf .eq. 2 ) then
          write( * , * ) 'quadratic representation'
          write( 2 , * ) 'quadratic representation'
          write( * , * )
      else
          write( * , * ) 'cubic representation'
          write( 2 , * ) 'cubic representation'
          write( * , * )
      endif

      time( 1 ) = ( tinit / tmstep )
      densty = densty / dscale
      rinit = 0.1651d0

      if( ndatyp .eq. 1 ) then
          rinit = rexp
      endif

      vinit = ( 4.0 / 3.0 ) * pi * rinit ** 3
```

```
      write( * , * ) 'original timestep = ' , tmstep , 's'
      write( 2 , * ) 'original timestep = ' , tmstep , 's'
      write( * , * ) 'initial time = ' , tinit , 's'
      write( * , * ) 'initial radius = ' , rinit,'m'
      write( * , * ) 'initial volume = ' , vinit,'m^3'
      write( * , * ) 'inception pressure = ' , pincep , 'pa'
      write( * , * ) 'inception z-coord. = ' , dincep,'m'
      write( * , * ) 'vapour pressure = ' , pvap , 'pa'
      write( * , * ) 'maximum radius = ' , rmaxi,'m'
      write( * , * ) 'density = ' , densty , 'kg/m^3'
      write( * , * ) 'gamma (assumed) = ' , gamma
      write( * , * ) 'length scale = ' , xscale
      write( * , * ) 'pressure scale = ' , pscale
      write( * , * ) 'time scale = ' , tscale
      write( * , * ) 'velocity scale = ' , vnscal
      write( * , * ) 'energy scale = ' , escale
      write( * , * ) 'volume scale = ' , vscale
      write( * , * ) 'density scale = ' , dscale
      write( * , * ) 'vel. potential scale = ' , vpscal
      write( * , * ) 'dphi/dxi scale = ' , dxscal
      write( * , * ) 'dphi/dt scale = ' , dtscal
      write( * , * ) 'rddot scale = ' , rdscal

      tmstep = tmstep / tscale

      write( * , * ) 'scaled timestep = ' , tmstep
      write( 2 , * ) 'scaled timestep = ' , tmstep
      write( * , * )

      write( 2 , * ) 'initial time = ' , tinit , 's'
      write( 2 , * ) 'initial radius = ' , rinit,'m'
      write( 2 , * ) 'initial volume = ' , vinit,'m^3'
      write( 2 , * ) 'inception pressure = ' , pincep , 'pa'
      write( 2 , * ) 'inception z-coord. = ' , dincep,'m'
      write( 2 , * ) 'vapour pressure = ' , pvap , 'pa'
      write( 2 , * ) 'maximum radius = ' , rmaxi,'m'
      write( 2 , * ) 'density = ' , densty , 'kg/m^3'
      write( 2 , * ) 'gamma (assumed) = ' , gamma
      write( 2 , * ) 'length scale = ' , xscale
      write( 2 , * ) 'pressure scale = ' , pscale
      write( 2 , * ) 'time scale = ' , tscale
      write( 2 , * ) 'velocity scale = ' , vnscal
      write( 2 , * ) 'energy scale = ' , escale
```

```
      write( 2 , * ) 'volume scale = ' , vscale
      write( 2 , * ) 'density scale = ' , dscale
      write( 2 , * ) 'vel. potential scale = ' , vpscal
      write( 2 , * ) 'dphi/dxi scale = ' , dxscal
      write( 2 , * ) 'dphi/dt scale = ' , dtscal
      write( 2 , * ) 'rddot scale = ' , rdscal

      dincep = dincep / xscale
      vinit = vinit / vscale
      close( 2 )


c  Set up the time array
c  ---------------------

      time( 1 ) = time( 1 ) * tmstep

      do 80 i = 2 , numfls , 1

          time( i ) = time( i - 1 ) + tmstep

 80   continue


c  Housekeeping
c  ------------

 500  format( a11 )
 510  format( 1x , 4f14.7 )
 520  format( 1x , 2f14.7 )
 2000 format( i8 )

      return
      end
```

```
c      ****************************************************************
c      *                        Subroutine PROJFW                     *
c      ****************************************************************
c      *                                                              *
c      * PROJFW solves the equation                                   *
c      *  _   _                                                       *
c      *  t . a = 0                                                   *
c      *                                                              *
c      * i.e.,  ( r(xi) - rf ) tr  +  ( z(xi) - zf ) tz  = 0         *
c      *                                                              *
c      * using the secant method,                                     *
c      *           _                          dr      dz              *
c      *     where t = ( tr , tz ) = ( ----- , ----- )                *
c      *                                    dxi     dxi               *
c      *                                                              *
c      * i.e., when the dot product is zero the forward normal        *
c      *       displacement is found                                  *
c      *                                                              *
c      * It is based on code written by J Best, but modified for      *
c      * use in this work.                                            *
c      *                                                              *
c      ****************************************************************

c      Local variable list
c      -------------------

c      acc       = test condition for iterative loop          dbl prec
c      displ     = forward normal displacement                dbl prec
c      dot       = dot product                                dbl prec
c      fe        = function value at end of interval          dbl prec
c      fi        = function value at start of interval        dbl prec
c      fn        = function value at solution point           dbl prec
c      jn        = solution node                              integer
c      mp        = dimensioning variable for arrays           integer
c      nerr      = 1 if root not found                        integer
c      nit       = iteration limit                            integer
c      r1        = coord. of estimated solution point         dbl prec
c      ra1       = spline parameter (next)                    dbl prec
c      rb1       = spline parameter (next)                    dbl prec
c      rc1       = spline parameter (next)                    dbl prec
c      rd1       = spline parameter (next)                    dbl prec
c      re        = coord. of end ((j)th) node (next)          dbl prec
c      rf        = coord. of reference node (present)         dbl prec
```

```
c       ri        = coord. of initial ((j-1)th) node (next)  dbl prec
c       rn        = coord.of solution point (next)           dbl prec
c       rr        = dummy variable                           dbl prec
c       se        = end of interval                          dbl prec
c       sgni      = sign of function value at start of int.  dbl prec
c       sgnn      = sign of function value at solution point dbl prec
c       si        = start of interval                        dbl prec
c       sn        = arclength of solution point              dbl prec
c       ss        = dummy arclength variable                 dbl prec
c       z1        = coord. of estimated solution point       dbl prec
c       za1       = spline parameter (next)                  dbl prec
c       zb1       = spline parameter (next)                  dbl prec
c       zc1       = spline parameter (next)                  dbl prec
c       zd1       = spline parameter (next)                  dbl prec
c       ze        = coord. of end ((j)th) node (next)        dbl prec
c       zf        = coord. of reference node (present)       dbl prec
c       zi        = coord. of initial ((j-1)th) node (next)  dbl prec
c       zn        = coord. of solution point (next)          dbl prec
c       zz        = dummy variable                           dbl prec

c last update 13/7/93  sbh


        subroutine projfw( ra , rb , rc , rd ,
     &                     za , zb , zc , zd ,
     &                     s , rf , zf , tr , tz , mp , displ ,
     &                     nnodes , sn , j , jn , r1 , z1 , nerr )


c  Dimension variables
c  -------------------

        implicit double precision( a - h , o - z )

        parameter( acc = 1.0d-12 , nit = 25 )

        dimension ra( 0 : mp ) , za( 0 : mp ) ,
     &            rb( 0 : mp ) , zb( 0 : mp ) ,
     &            rc( 0 : mp ) , zc( 0 : mp ) ,
     &            rd( 0 : mp ) , zd( 0 : mp ) ,
     &            s( 0 : mp )
```

```
c  Set constants
c  -------------

       nerr = 0


c  Main iteration loop for secant method
c  -------------------------------------

       do 10 j = 1 , nnodes , 1

           jm1 = j - 1
           ss = s( j )

           ra1 = ra( jm1 )
           rb1 = rb( jm1 )
           rc1 = rc( jm1 )
           rd1 = rd( jm1 )
           za1 = za( jm1 )
           zb1 = zb( jm1 )
           zc1 = zc( jm1 )
           zd1 = zd( jm1 )

           ri = ra1
           zi = za1
           fi = ( ri - rf ) * tr + ( zi - zf ) * tz

           re = ra( j )
           ze = za( j )
           fe = ( re - rf ) *tr + ( ze - zf ) *tz

           if( abs( fi ) . lt . acc ) then
               sn = 0.0
               rn = ri
               zn = zi
               goto 4
           endif

           if( abs( fe ) . lt . acc ) then
               sn = ss
               rn = re
               zn = ze
               goto 4
           endif
```

```
            sgni = fi / abs( fi )
            sgne = fe / abs( fe )

            if( sgni * sgne . lt . 0.0 ) then

c            There is a stationary point within the interval.
c            ------------------------------------------------
c            It is found assuming there is only one such point.
c            --------------------------------------------------
             si = 0.0
             se = ss

            do 5 i = 1 , nit , 1

                 sn = ( fi * se - fe * si ) / ( fi - fe )
                 rn = ra1 + sn * ( rb1 + sn * ( rc1 + sn * rd1 ) )
                 zn = za1 + sn * ( zb1 + sn * ( zc1 + sn * zd1 ) )

                 fn = ( rn - rf ) *tr + ( zn - zf ) * tz

                 if( abs( fn ) . lt . acc ) goto 4

                 sgnn = fn / abs( fn )

                 if( sgni * sgnn . gt . 0.0 ) then

c                 initial and end points have the same signs
c                 ------------------------------------------
c                 (root is not in interval)
c                 -------------------------
                  si = sn
                  fi = fn
                  ri = rn
                  zi = zn

                 else
```

```
c                initial and end points have opposite signs
c                ------------------------------------------
c                (root is in interval)
c                --------------------
                 se = sn
                 fe = fn
                 re = rn
                 ze = zn

                endif

 5           continue

            endif

 10     continue


c  Root not found - estimate displ using internodal distance
c  ---------------------------------------------------------

        nerr = 1
        rn = r1
        zn = z1
        sn = s( jn )
        j = jn

 4      rr = rn - rf
        zz = zn - zf


c  Root found - compute displ and dot product
c  ------------------------------------------

        displ = ( rr**2.0 + zz**2.0 ) ** 0.5
        dot = -rr * tz + zz *tr

        if( dot . lt . 0.0 ) then

c           bubble is contracting
c           ---------------------
            displ = displ
```

```
        else

c           bubble is expanding
c           -------------------
            displ = -1.0 * displ

        endif


c  Housekeeping
c  ------------

        return
        end
```

```
c       ****************************************************************
c       *                          Function DIST                       *
c       ****************************************************************
c       *                                                              *
c       * DIST computes the Euclidean distance between two points.     *
c       *                                                              *
c       ****************************************************************

c       Local variable list
c       -------------------

c       r1        = radial coordinate of point no. 1          dbl prec
c       r2        = radial coordinate of point no. 2          dbl prec
c       z1        = vertical coordinate of point no. 1        dbl prec
c       z2        = vertical coordinate of point no. 2        dbl prec
c       dist      = Euclidean distance between points 1 & 2   dbl prec

c  last update 10/4/93


        function dist( r1 , z1 , r2 , z2 )


c  Initialise variables
c  --------------------

        implicit double precision( a - h , o - z )


c  Start of function
c  -----------------

        dist = ( ( r1 - r2 ) ** 2.0 + ( z1 - z2 ) ** 2.0 ) ** 0.5


c  Housekeeping
c  ------------

        return
        end
```

```
c     ****************************************************************
c     *                      Subroutine PROJBK                       *
c     ****************************************************************
c     *                                                              *
c     * PROJBK solves the equation                                   *
c     *                                                              *
c     *      2                                                       *
c     *   d(d )                                                      *
c     *   -----  = 0                                                 *
c     *    dxi                                                       *
c     *                                                              *
c     * i.e.,                                                        *
c     *                                                              *
c     *                  dr(xi)                    dz(xi)            *
c     *   ( r(xi) - rf ) ------  +  ( z(xi) - zf ) ------  = 0       *
c     *                   dxi                       dxi              *
c     *                                                              *
c     * using the secant method.                                     *
c     *                                                              *
c     *                2                                             *
c     *     i.e., when d  is minimum then the normal is found.       *
c     *                                                              *
c     * It is based on code written by J Best, but modified for      *
c     * use in this work.                                            *
c     *                                                              *
c     ****************************************************************

c     Local variable list
c     -------------------

c     acc       = test condition for iterative loop          dbl prec
c     de        = distance from final to reference node      dbl prec
c     di        = distance from initial to reference node    dbl prec
c     displ     = backward normal displacement               dbl prec
c     dmin      = distance from node 0 to reference node     dbl prec
c     dot       = dot product                                dbl prec
c     dr1       = dr1/dxi                                    dbl prec
c     ds        = distance to the solution point             dbl prec
c     dz1       = dz1/dxi                                    dbl prec
c     drdt      = used to find normal vector components      dbl prec
c     dsdt      = used to find normal vector components      dbl prec
c     dzdt      = used to find normal vector components      dbl prec
c     fe        = function value at end of interval          dbl prec
c     fi        = function value at start of interval        dbl prec
```

```
c       fn        = function value at solution point          dbl prec
c       jdum      = dummy variable                            dbl prec
c       mki       = dummy variable                            integer
c       mp        = dimensioning variable for arrays          integer
c       r(0:,)    = radial coordinate array - (i-1)th surf.   dbl prec
c       r1        = radial coordinate of (j-1)th node (prev)  dbl prec
c       ra1       = spline parameter (previous)               dbl prec
c       rb1       = spline parameter (previous)               dbl prec
c       rc1       = spline parameter (previous)               dbl prec
c       rd1       = spline parameter (previous)               dbl prec
c       re        = coord. of end ((j)th) node (prev)         dbl prec
c       rf        = coord. of reference node (present)        dbl prec
c       rf1       = dummy variable                            dbl prec
c       ri        = coord. of initial ((j-1)th) node (prev)   dbl prec
c       rmin      = radial coordinate of solution point       dbl prec
c       rn        = radial component of normal vector         dbl prec
c       rr        = dummy variable                            dbl prec
c       rs        = radial coordinate of solution point (prv)dbl prec
c       se        = end of interval                           dbl prec
c       si        = start of interval                         dbl prec
c       sn        = arclength of solution point               dbl prec
c       ss        = dummy arclength variable                  dbl prec
c       sgne      = sign of function value at end of int.     dbl prec
c       sgni      = sign of function value at start of int.   dbl prec
c       sgnn      = sign of function value at solution point  dbl prec
c       smin      = arclength of solution point               dbl prec
c       ss        = arclength at node j                       dbl prec
c       x         = dummy variable                            dbl prec
c       z(0:,)    = vertical coordinate array - (i-1)th surf.dbl prec
c       z1        = vertical coordinate of (j-1)th node (prv)dbl prec
c       za1       = spline parameter (previous)               dbl prec
c       zb1       = spline parameter (previous)               dbl prec
c       zc1       = spline parameter (previous)               dbl prec
c       zd1       = spline parameter (previous)               dbl prec
c       ze        = coord. of end ((j)th) node (prev)         dbl prec
c       zf        = coord. of reference node (present)        dbl prec
c       zf1       = dummy variable                            dbl prec
c       zi        = coord. of initial ((j-1)th) node (prev)   dbl prec
c       zmin      = vertical coordinate of solution point     dbl prec
c       zn        = vertical coordinate of normal vector      dbl prec
c       zs        = vert. coordinate of solution point (prev)dbl prec
c       zz        = dummy variable                            dbl prec

c last update 13/7/93  sbh
```

```
      subroutine projbk( ra , rb , rc , rd ,
     &                   za , zb , zc , zd ,
     &                   r , z ,
     &                   s , rf , zf , mp , displ ,
     &                   sn , jdum , rs , zs , nnodes)


c  Dimension variables
c  -------------------

        implicit double precision( a - h , o - z )

        parameter( acc = 1.0d-12 )

        dimension r( 0 : mp ) , z( 0 : mp ) ,
     &          ra( 0 : mp ) , za( 0 : mp ) ,
     &          rb( 0 : mp ) , zb( 0 : mp ) ,
     &          rc( 0 : mp ) , zc( 0 : mp ) ,
     &          rd( 0 : mp ) , zd( 0 : mp ) ,
     &          s( 0 : mp )


c  Set constants
c  -------------

        rf1 = rf
        zf1 = zf
        rmin = r( 0 )
        zmin = z( 0 )
        dmin = dist( rmin , zmin , rf , zf )
        smin = 0.0
        mki = 0


c  Main iteration loop for secant method
c  -------------------------------------

        do 2 j = 1 , nnodes , 1

            jm1 = j - 1
            ss = s( j )
```

```
            ri = r( jm1 )
            zi = z( jm1 )
            re = r( j )
            ze = z( j )
            di = dist( ri , zi , rf , zf )
            de = dist( re , ze , rf , zf )

            if( di . lt . dmin ) then

c               if the distance from initial to ref < dmin
c               ------------------------------------------
                dmin = di
                rmin = ri
                zmin = zi
                smin = 0.0
                mki = jm1

             endif


            if( de . lt . dmin ) then

c               if the distance from end to ref < dmin
c               --------------------------------------
                dmin = de
                rmin = re
                zmin = ze
                smin = ss
                mki = jm1

            endif

            ra1 = ra( jm1 )
            rb1 = rb( jm1 )
            rc1 = rc( jm1 )
            rd1 = rd( jm1 )

            za1 = za( jm1 )
            zb1 = zb( jm1 )
            zc1 = zc( jm1 )
            zd1 = zd( jm1 )
```

```
      x = 0.0
      r1 = ra1 + x * ( rb1 + x * ( rc1 + x * rd1 ) )
      z1 = za1 + x * ( zb1 + x * ( zc1 + x * zd1 ) )
      dr1 = rb1 + x * ( 2.0 * rc1 + 3.0 * x * rd1 )
      dz1 = zb1 + x * ( 2.0 * zc1 + 3.0 * x * zd1 )
      fi = ( r1 - rf1 ) * dr1 + ( z1 - zf1 ) * dz1

      x = ss
      r1 = ra1 + x * ( rb1 + x * ( rc1 + x * rd1 ) )
      z1 = za1 + x * ( zb1 + x * ( zc1 + x * zd1 ) )
      dr1 = rb1 + x * ( 2.0 * rc1 + 3.0 * x * rd1 )
      dz1 = zb1 + x * ( 2.0 * zc1 + 3.0 * x * zd1 )
      fe = ( r1 - rf1 ) * dr1 + ( z1 - zf1 ) * dz1

      if( abs( fi ) . lt . acc ) then
          sn = 0.0
          goto 4
      endif

      if( abs( fe ) . lt . acc ) then
          sn = ss
          goto 4
      endif

      sgni = fi / abs( fi )
      sgne = fe / abs( fe )

      if( sgni * sgne . lt . 0.0 ) then

c      There is a stationary point within the interval.
c      ------------------------------------------------
c      It is found assuming there is only one such point.
c      --------------------------------------------------
      si = 0.0
      se = ss
3     sn = ( fi * se - fe * si ) / ( fi - fe )

      x = sn
      r1 = ra1 + x * ( rb1 + x * ( rc1 + x * rd1 ) )
      z1 = za1 + x * ( zb1 + x * ( zc1 + x * zd1 ) )
      dr1 = rb1 + x * ( 2.0 * rc1 + 3.0 * x * rd1 )
      dz1 = zb1 + x * ( 2.0 * zc1 + 3.0 * x * zd1 )
      fn = ( r1 - rf1 ) * dr1 + ( z1 - zf1 ) * dz1
```

```
            if( abs( fn ) . lt . acc ) goto 4

c            point found
c            -----------

             sgnn = fn / abs( fn )

            if( sgni * sgnn . gt . 0.0 ) then

c            initial and end points have the same signs
c            ------------------------------------------
c            (root is not in interval)
c            -------------------------
             si = sn
             fi = fn

            else

c            initial and end points have opposite signs
c            ------------------------------------------
c            (root is in interval)
c            ---------------------
             se = sn
             fe = fn

             endif

             goto 3

            endif

            goto 2


c  Root found
c  ----------

 4          rs = ra1 + sn * ( rb1 + sn * ( rc1 + sn * rd1 ) )
            zs = za1 + sn * ( zb1 + sn * ( zc1 + sn * zd1 ) )
            ds = dist( rs , zs , rf , zf )
            jdum = j
```

```
            if( ds . lt . dmin ) then
             dmin = ds
             rmin = rs
             zmin = zs
             smin = sn
             mki = jm1
            endif

  2     continue

c  Compute normal vector components
c  --------------------------------

        dzdt =  zb( mki ) + smin * ( 2.0 * zc( mki ) +
     &        3.0 * smin * zd( mki ) )
        drdt = rb( mki ) + smin * ( 2.0 * rc( mki ) +
     &         3.0 * smin * rd( mki ) )
        dsdt = ( drdt ** 2.0 + dzdt ** 2.0 ) ** 0.5
        rn = - dzdt / dsdt
        zn = drdt / dsdt

c  Compute displ and dot product
c  -----------------------------

        rr = rmin - rf
        zz = zmin - zf
        dot = rr * rn + zz * zn

        if( dot . lt . 0.0 ) then
c           bubble is expanding
c           --------------------

            displ = -1.0 * dmin
        else
c           bubble is contracting
c           ---------------------

            displ = dmin
        endif

c  Housekeeping
c  ------------
       return
       end
```

```
c       ***************************************************************
c       *                        Subroutine DPHDT                     *
c       ***************************************************************
c       *                                                             *
c       * DPHDT computes the time rate of change of the velocity      *
c       * potential.                                                  *
c       *                                                             *
c       ***************************************************************

c  last update 10/4/93  sbh


        subroutine dphdt( dphidt , phitf , phitb , tmstep , ndphdt ,
     &                    numfls , maxfil , nnodes , maxnod ,
     &                    phiacc )


c  Dimension variables
c  -------------------

        implicit double precision ( a - h , o - z )

        dimension phitf( 0 : maxnod , maxfil ) ,
     &            dphidt( 0 : maxnod , maxfil ) ,
     &            phitb( 0 : maxnod , maxfil )


c  Use finite difference approximation to find dphidt
c  --------------------------------------------------

        do 30 j = 0 , nnodes , 1

            do 30 i = 3 , numfls - 2 , 1

                dphidt( j , i ) = ( phitf( j , i ) -
     &             phitb( j , i ) ) / ( 2.0d0 * tmstep )

 30     continue


c  Housekeeping
c  ------------
        return
        end
```

```
c       ***************************************************************
c       *                        Subroutine DPHDZ                     *
c       ***************************************************************
c       *                                                             *
c       * DPHDZ computes the tangential velocity dphi/dxi.            *
c       *                                                             *
c       ***************************************************************


c       Local variable list
c       -------------------

c       xij       = arclength difference nodes j to j-1       dbl prec
c       xip1      = arclength difference nodes j+1 to j       dbl prec

c  last update 10/4/93


        subroutine dphdz( dphidz , t , phic ,
     &                    numfls , maxfil , nnodes , maxnod )


c  Dimension variables
c  -------------------

        implicit double precision ( a - h , o - z )

        dimension phic( 0 : maxnod , maxfil ) ,
     &            dphidz( 0 : maxnod , maxfil ) ,
     &            t( 0 : maxnod , maxfil )


c  Fit a local quadratic function to the phic values and
c  -----------------------------------------------------
c  differentiate w.r.t. xi to find dphidxi
c  ---------------------------------------

        do 20 i = 1 , numfls , 1

            do 30 j = 1 , nnodes - 1, 1

                xij = t( j , i ) - t( j - 1 , i )
                xijp1 = t( j + 1 , i ) - t( j , i )
```

```
                dphidz( j , i ) = ( xij ** 2.0d0 *
     &              phic( j + 1, i ) - ( xij ** 2.0d0 -
     &              xijp1 ** 2.0d0 ) * phic( j , i ) -
     &              xijp1 ** 2.0d0 * phic( j - 1 , i ) )
     &              / ( xij * xijp1 * ( xij + xijp1 ) )

30          continue

            dphidz( 0 , i ) = 0.0d0
            dphidz( nnodes , i ) = 0.0d0

20      continue


c   Housekeeping
c   ------------

        return
        end
```

```
c       ************************************************************
c       *                      Subroutine FORTRJ                   *
c       ************************************************************
c       *                                                          *
c       * FORTRJ solves the equation                               *
c       *     _*  _                                                *
c       *     u . a = 0                                            *
c       *                                                          *
c       * using the secant method,                                 *
c       *       _*                                  _              *
c       * where u  is the normal to the velocity vector u, in order *
c       * to compute the forward trajectory of the node P.         *
c       *                                                          *
c       * It is based on code written by J Best, but modified for  *
c       * use in this work.                                        *
c       *                                                          *
c       ************************************************************

c       Local variable list
c       -------------------

c       acc      = test condition for iterative loop        dbl prec
c       fe       = function value at end of interval        dbl prec
c       fi       = function value at start of interval      dbl prec
c       fn       = function value at solution point         dbl prec
c       jm1      = dummy variable                           integer
c       mp       = dimensioning variable for arrays         integer
c       nerr     = 1 if root not found                      integer
c       node     = dummy variable                           integer
c       phix     = velocity potential at solution point     dbl prec
c       ra1      = spline parameter (next)                  dbl prec
c       rb1      = spline parameter (next)                  dbl prec
c       rc1      = spline parameter (next)                  dbl prec
c       rd1      = spline parameter (next)                  dbl prec
c       re       = coord. of end ((j)th) node (next)        dbl prec
c       rf       = coord. of reference node (present)       dbl prec
c       ri       = coord. of initial ((j-1)th) node (next)  dbl prec
c       rn       = coord.of solution point (next)           dbl prec
c       se       = end of interval                          dbl prec
c       sgne     = sign of function value at end of int.    dbl prec
c       sgni     = sign of function value at start of int.  dbl prec
c       sgnn     = sign of function value at solution point dbl prec
c       si       = start of interval                        dbl prec
```

```
c       sn        = arclength of solution point              dbl prec
c       ss        = dummy arclength variable                 dbl prec
c       tr        = radial tangential component              dbl prec
c       tz        = vertical tangential component            dbl prec
c       za1       = spline parameter (next)                  dbl prec
c       zb1       = spline parameter (next)                  dbl prec
c       zc1       = spline parameter (next)                  dbl prec
c       zd1       = spline parameter (next)                  dbl prec
c       ze        = coord. of end ((j)th) node (next)        dbl prec
c       zf        = coord. of reference node (present)       dbl prec
c       zi        = coord. of initial ((j-1)th) node (next)  dbl prec
c       zn        = coord. of solution point (next)          dbl prec

c  last update 26/8/93  sbh


        subroutine fortrj( ra , rb ,
     &                     rc , rd ,
     &                     za , zb ,
     &                     zc , zd ,
     &                     s , rf , zf , phic ,
     &                     tr , tz , mp , phix , node ,
     &                     sn , fn , j , nerr , nnodes )


c  Dimension variables
c  -------------------

        implicit double precision( a - h , o - z )

        parameter( acc = 1.0d-12 )

        dimension ra( 0 : mp ) , za( 0 : mp ) ,
     &            rb( 0 : mp ) , zb( 0 : mp ) ,
     &            rc( 0 : mp ) , zc( 0 : mp ) ,
     &            rd( 0 : mp ) , zd( 0 : mp ) ,
     &            s( 0 : mp ) , phic( 0 : mp )


c  Set constants
c  -------------

        nerr = 0
        j = 0
```

```
c  Main iteration loop for secant method
c  ------------------------------------

 2      j = j + 1

        jm1 = j - 1
        ss = s( j )

        ra1 = ra( jm1 )
        rb1 = rb( jm1 )
        rc1 = rc( jm1 )
        rd1 = rd( jm1 )
        za1 = za( jm1 )
        zb1 = zb( jm1 )
        zc1 = zc( jm1 )
        zd1 = zd( jm1 )

        ri = ra1
        zi = za1
        fi = ( ri - rf ) * tr + ( zi - zf ) * tz

        re = ra( j )
        ze = za( j )
        fe = ( re - rf ) *tr + ( ze - zf ) *tz

        if( abs( fi ) . lt . acc ) then
            sn = 0.0
            rn = ri
            zn = zi
            goto 4
        endif

        if( abs( fe ) . lt . acc ) then
            sn = ss
            rn = re
            zn = ze
            goto 4
        endif

        sgni = fi / abs( fi )
        sgne = fe / abs( fe )

        if( sgni * sgne . lt . 0.0 ) then
```

```
c           There is a stationary point within the interval.
c           ------------------------------------------------
c           It is found assuming there is only one such point.
c           --------------------------------------------------
            si = 0.0
            se = ss

  3         sn = ( fi * se - fe * si ) / ( fi - fe )
            rn = ra1 + sn * ( rb1 + sn * ( rc1 + sn * rd1 ) )
            zn = za1 + sn * ( zb1 + sn * ( zc1 + sn * zd1 ) )
            fn = ( rn - rf ) * tr + ( zn - zf ) * tz

            if( abs( fn ) . lt . acc ) goto 4
c           root found
c           ----------

            sgnn = fn / abs( fn )

            if( sgni * sgnn . gt . 0.0 ) then

c               initial and end points have the same signs
c               ------------------------------------------
c               (root is not in interval)
c               -------------------------
                si = sn
                fi = fn
                ri = rn
                zi = zn

            else

c               initial and end points have opposite signs
c               ------------------------------------------
c               (root is in interval)
c               ---------------------
                se = sn
                fe = fn
                re = rn
                ze = zn

            endif

            goto 3
```

```
        endif

        if( j .eq. nnodes ) then

c           root not found - use (i+1,j)th nodal value instead
c           ---------------------------------------------------
            nerr = 1
            phix = phic( jm1 )
            goto 5

        endif

        goto 2


c  Root found
c  ----------

 4      phix = phic( jm1 ) + sn * ( phic( j ) - phic( jm1 ) )
     &         / s( j )

 5      continue


c  Housekeeping
c  ------------

        return
        end
```

```
c       ************************************************************
c       *                      Subroutine BAKTRJ                   *
c       ************************************************************
c       *                                                          *
c       * BAKTRJ solves the equation                               *
c       *     _*  _                                                *
c       *     u . a = 0                                            *
c       *                                                          *
c       * using the bisection method,                              *
c       *       _*                                  _              *
c       * where u  is the normal to the velocity vector u, in order *
c       * to compute the forward trajectory of the node P.         *
c       *                                                          *
c       * It is based on code written by J Best, but modified for  *
c       * use in this work.                                        *
c       *                                                          *
c       ************************************************************


c       Local variable list
c       -------------------

c       acc      = test condition for iterative loop        dbl prec
c       dpdzn    = dphi/dxi interpolated value              dbl prec
c       dpn      = dphi/dn interpolated value               dbl prec
c       drn      = dr/dxi                                   dbl prec
c       dzn      = dz/dxi                                   dbl prec
c       fl       = function value at start of interval      dbl prec
c       fu       = function value at end of interval        dbl prec
c       f        = current function value                   dbl prec
c       im1      = dummy variable                           dbl prec
c       mp       = dimensioning variable for arrays         integer
c       nerr     = 1 if root not found                      integer
c       phix     = velocity potential at solution point     dbl prec
c       r(0:,)   = radial coordinate array - (i-1)th surf.  dbl prec
c       ra1      = spline parameter (previous)              dbl prec
c       rb1      = spline parameter (previous)              dbl prec
c       rc1      = spline parameter (previous)              dbl prec
c       rd1      = spline parameter (previous)              dbl prec
c       rn       = radial coordinate of solution point      dbl prec
c       xil      = arclength at start of interval           dbl prec
c       xiu      = arclength at end of interval             dbl prec
c       xin      = arclength of solution point              dbl prec
c       z(0:,)   = vertical coordinate array - (i-1)th surf.dbl prec
```

```
c       za1       = spline parameter (previous)              dbl prec
c       zb1       = spline parameter (previous)              dbl prec
c       zc1       = spline parameter (previous)              dbl prec
c       zd1       = spline parameter (previous)              dbl prec
c       zn        = vertical coordinate of solution point    dbl prec

c last update 13/7/93  sbh


        subroutine baktrj( ra , rb , rc , rd ,
     &                     za , zb ,  zc , zd  ,
     &                     r , z ,
     &                     dphidn , phic ,
     &                     dphidz , s , rf , zf ,
     &                     mp , nnodes , phix , nerr )


c  Dimension variables
c  -------------------

        implicit double precision( a - h , o - z )

        parameter( acc = 1.0d-12 )

        dimension ra( 0 : mp ) , za( 0 : mp ) ,
     &            rb( 0 : mp ) , zb( 0 : mp ) ,
     &            rc( 0 : mp ) , zc( 0 : mp ) ,
     &            rd( 0 : mp ) , zd( 0 : mp ) ,
     &            s( 0 : mp ) , phic( 0 : mp ) ,
     &            dphidn( 0 : mp ) ,
     &            dphidz( 0 : mp ) , r( 0 : mp ) ,
     &            z( 0 : mp )


c  Set constants
c  -------------

        nerr = 0
        i = -1
```

```
c  Main iteration loop for bisection method
c  ----------------------------------------

 1      i = i + 1

        f = -1.0d0 * ( dphidn( i ) * rb( i ) +
     &                 dphidz( i ) * zb( i ) ) *
     &               ( r( i ) - rf )
     &      +        ( - 1.0d0 * dphidn( i ) * zb( i ) +
     &                          dphidz( i ) * rb( i ) ) *
     &                        ( z( i ) - zf )


        if ( abs( f ) .lt. acc ) then

c           root found
c           ----------
            i = i + 1
            xin = 0.0d0
            im1 = i - 1
            goto 5

        endif

        if( i .eq. 0 ) then

            fu = f

        endif

        if( i .ge. 1 ) then

            fl = fu
            fu = f
            if( fl * fu .lt. 0.0d0 ) goto 2

        endif

        if( i .eq. nnodes ) then
```

```
c           root not found
c           --------------
            nerr = 1
            phix = phic( i - 1 )
            goto 6

        endif

        goto 1

 2      im1 = i - 1
        xil = 0.0
        xiu = s( i )

        ra1 = ra( im1 )
        rb1 = rb( im1 )
        rc1 = rc( im1 )
        rd1 = rd( im1 )

        za1 = za( im1 )
        zb1 = zb( im1 )
        zc1 = zc( im1 )
        zd1 = zd( im1 )

 3      xin = 0.5d0 * ( xil + xiu )
        rn = ra1 + xin * ( rb1 + xin * ( rc1 + xin * rd1 ) )
        zn = za1 + xin * ( zb1 + xin * ( zc1 + xin * zd1 ) )
        drn = rb1 + xin * ( 2.0d0 * rc1 + 3.0d0 * xin * rd1 )
        dzn = zb1 + xin * ( 2.0d0 * zc1 + 3.0d0 * xin * zd1 )

        dpdzn = dphidz( im1 ) + xin *
     &          ( dphidz( i ) - dphidz( im1 ) ) / s( i )
        dpn   = dphidn( im1 ) + xin *
     &          ( dphidn( i ) - dphidn( im1 ) ) / s( i )

        f = -1.0d0 * ( dpn * drn + dpdzn * dzn ) *
     &               ( rn - rf )
     &     +         ( -1.0d0 * dpn * dzn + dpdzn * drn ) *
     &               ( zn - zf )

        if( abs( f ) .lt. acc ) goto 5
c           root found
c           ----------
```

```
        if( fl * f .le. 0.0d0 ) then

c           initial and end points have opposite signs.
c           -------------------------------------------
c           reset upper limit to xin.
c           -------------------------
            fu = f
            xiu = xin

        else

c           initial and end points have the same signs.
c           -------------------------------------------
c           reset lower limit to xin.
c           -------------------------
            fl = f
            xil = xin

        endif

        goto 3


c       Root found - interpolate linearly
c       ---------------------------------

 5      phix = phic( im1 ) + xin * ( phic( i ) - phic( im1 ) )
     &         / s( i )

 6      continue


c  Housekeeping
c  ------------

        return
        end
```

```
c       ************************************************************
c       *                      Subroutine VOLUML                   *
c       ************************************************************
c       *                                                          *
c       * VOLUML computes the volume of a bubble having a linear   *
c       * surface representation.                                  *
c       *                                                          *
c       ************************************************************

c       Local variable list
c       -------------------

c       area      = cross sectional area of bubble          dbl prec
c       mp        = dimensioning variable for arrays        integer
c       np        = total no. surface nodes - 1             integer
c       pi        = trigonometric pi                        dbl prec
c       ra()      = spline parameter array                  dbl prec
c       vol       = bubble volume                           dbl prec
c       za()      = spline parameter array                  dbl prec

c  last update 27/10/93  sbh


        subroutine voluml( mp , np , ra , za , vol )


c  Dimension variables
c  -------------------

        implicit double precision( a - h , o - z )

        dimension  ra( 0 : mp ) , za( 0 : mp )


c  Set constants
c  -------------

        pi = 4.0 * atan( 1.0 )
        area = 0.0d0
        vol = 0.0d0
```

```
c  Compute volume
c  --------------

        do 1 i = 1 , np, 1

            area = area + abs( za(i) - za(i-1) ) * ( min( ra(i-1) , ra(i) )
     &              + 0.5d0 * abs( ra(i) - ra(i-1) ) )

 1      continue

        vol = 2.0d0 * pi * area


c  Housekeeping
c  ------------

        return
        end
```

```
c       ************************************************************
c       *                      Subroutine VOLUMC                   *
c       ************************************************************
c       *                                                          *
c       * VOLUML computes the volume of a bubble having a cubic    *
c       * surface representation.                                  *
c       *                                                          *
c       * It is based on code written by J Best, but modified for  *
c       * use in this work.                                        *
c       *                                                          *
c       ************************************************************

c       Local variable list
c       -------------------

c       a()       = dummy array                                  dbl prec
c       b()       = dummy array                                  dbl prec
c       dd        = dummy variable                               dbl prec
c       ex        = dummy real variable                          dbl prec
c       help      = dummy variable                               dbl prec
c       mp        = dimensioning variable for arrays             integer
c       np        = total no. surface nodes - 1                  integer
c       pi        = trigonometric pi                             dbl prec
c       ra()      = spline parameter array                       dbl prec
c       rb()      = spline parameter array                       dbl prec
c       rc()      = spline parameter array                       dbl prec
c       rd()      = spline parameter array                       dbl prec
c       s()       = segment arclength array                      dbl prec
c       za()      = spline parameter array                       dbl prec
c       zb()      = spline parameter array                       dbl prec
c       zc()      = spline parameter array                       dbl prec
c       zd()      = spline parameter array                       dbl prec
c       volume    = bubble volume                                dbl prec

c  last update 27/10/93  sbh


      subroutine volumc( mp , np ,
     &                   ra , rb , rc , rd ,
     &                   za , zb , zc , zd ,
     &                   s , volume )
```

```
c  DImension variables
c  -------------------

       implicit double precision( a - h , o - z )

       dimension
     &         ra( 0 : mp ) , za( 0 : mp ) ,
     &         rb( 0 : mp ) , zb( 0 : mp ) ,
     &         rc( 0 : mp ) , zc( 0 : mp ) ,
     &         rd( 0 : mp ) , zd( 0 : mp ) ,
     &         s( 0 : mp ) , a( 7 ) , b( 9 )


c  Set constants
c  -------------

       pi = 4.0 * atan( 1.0 )
       volume = 0.0


c  Compute volume
c  --------------

       do 1 i = 0 , np - 1 , 1

           dd = s( i + 1 )
           a( 1 ) = ra( i ) * ra( i )
           a( 2 ) = 2.0 * ra( i ) * rb( i )
           a( 3 ) = 2.0 * ra( i ) * rc( i ) + rb( i ) * rb( i )
           a( 4 ) = 2.0 * ra( i ) * rd( i ) + 2.0 * rb( i ) * rc( i )
           a( 5 ) = 2.0 * rb( i ) * rd( i ) + rc( i ) * rc( i )
           a( 6 ) = 2.0 * rc( i ) * rd( i )
           a( 7 ) = rd( i ) * rd( i )

           do 3 k = 1 , 9 , 1

               ex = 1.0 * k
               b( k ) = dd ** ex / ex

 3         continue
```

```
            do 2 j = 1 , 7 , 1

                help = zb( i ) * b( j ) + 2.0 * zc( i ) * b( j + 1 ) +
     &                 3.0 * zd( i ) * b( j + 2 )
                volume = volume + a( j ) * help

 2          continue

 1      continue

        volume = pi * volume


c  Housekeeping
c  ------------

        return
        end
```

```
c       ************************************************************
c       *                      Subroutine ENERG                    *
c       ************************************************************
c       *                                                          *
c       * ENERG computes the kinetic energy of an evolving bubble. *
c       *                                                          *
c       * It is based on code written by J Best, but modified for  *
c       * use in this work.                                        *
c       *                                                          *
c       ************************************************************

c       Local variable list
c       -------------------

c       energy   = kinetic energy of bubble                 dbl prec
c       jg       = Max. no. points for Gaussian integration integer
c       jm1      = dummy variable                           integer
c       mp       = dimensioning variable for arrays         integer
c       np       = total no. surface nodes - 1              integer
c       p        = potential                                dbl prec
c       pi       = trigonometric pi                         dbl prec
c       pa()     = potential spline parameter array         dbl prec
c       pb()     = potential spline parameter array         dbl prec
c       pc()     = potential spline parameter array         dbl prec
c       pd()     = potential spline parameter array         dbl prec
c       q()      = dphi/dn array                            dbl prec
c       qm1      = dummy variable                           dbl prec
c       qq       = dummy variable                           dbl prec
c       r        = radial coordinate                        dbl prec
c       ra()     = coordinate spline parameter array        dbl prec
c       rb()     = coordinate spline parameter array        dbl prec
c       rc()     = coordinate spline parameter array        dbl prec
c       rd()     = coordinate spline parameter array        dbl prec
c       t()      = arclength parameter array                dbl prec
c       t0       = arclength at start of interval           dbl prec
c       t1       = arclength at end of interval             dbl prec
c       w()      = Gaussian weight array                    dbl prec
c       x()      = Gaussian integration array               dbl prec
c       y        = dummy variable                           dbl prec

c  last update 27/10/93  sbh
```

```
      subroutine energ( ra , rb , rc , rd ,
     &                  pa , pb , pc , pd ,
     &                  q , energy , t , mp , np )


c  Dimension variables
c  -------------------

      implicit double precision ( a - h , o - z )

      dimension
     &    ra( 0 : mp ) , rb( 0 : mp ) , rc( 0 : mp ) , rd( 0 : mp ) ,
     &    pa( 0 : mp ) , pb( 0 : mp ) , pc( 0 : mp ) , pd( 0 : mp ) ,
     &    q( 0 : mp ) , t( 0 : mp )

      parameter ( jg = 4 )

      dimension x( jg ) , w( jg )


c  Set constants
c  -------------

      pi = 4.0d0 * atan( 1.0d0 )
      energy = 0.0d0


c  Compute kinetic energy
c  ----------------------

      do 2 j = 1 , np , 1

          jm1 = j - 1
          t0 = t( jm1 )
          t1 = t( j )

          ra1 = ra( jm1 )
          rb1 = rb( jm1 )
          rc1 = rc( jm1 )
          rd1 = rd( jm1 )
```

```
            pa1 = pa( jm1 )
            pb1 = pb( jm1 )
            pc1 = pc( jm1 )
            pd1 = pd( jm1 )

            qm1 = -q( jm1 )
            q1 = -q( j )

            call dgauss( jg , t0 , t1 , x , w )

            do 1 k = 1 , jg , 1

                y  = x( k ) - t0
                r  = ra1 + y * ( rb1 + y * ( rc1 + rd1 * y ) )
                p  = pa1 + y * ( pb1 + y * ( pc1 + pd1 * y ) )
                qq = ( qm1 * ( t1 - x( k ) )  + q1 * y ) / ( t1 - t0 )
                energy = energy + w( k ) * r * p * qq

  1         continue

  2     continue

        energy =  pi * energy


c  Housekeeping
c  ------------

        return
        end
```

```
c       ************************************************************
c       *                      Subroutine LINEAR                   *
c       ************************************************************
c       *                                                          *
c       * LINEAR computes the directional arrays of the elements   *
c       * of a linearised surface.                                 *
c       *                                                          *
c       * It is based on an algorithm developed by J Best.         *
c       *                                                          *
c       ************************************************************

c       Local variable list
c       -------------------

c       mp        = dimensioning variable for arrays        integer
c       np        = total no. surface nodes - 1             integer
c       pi        = trigonometric pi                        dbl prec
c       r()       = radial coordinate array                 dbl prec
c       ra()      = spline parameter array                  dbl prec
c       rb()      = spline parameter array                  dbl prec
c       rc()      = spline parameter array                  dbl prec
c       rd()      = spline parameter array                  dbl prec
c       s()       = segment arclength array                 dbl prec
c       t()       = arclength parameter array               dbl prec
c       thjmh     = orientation of previous element         dbl prec
c       thjph     = orientation of next element             dbl prec
c       z()       = vertical coordinate array               dbl prec
c       za()      = spline parameter array                  dbl prec
c       zb()      = spline parameter array                  dbl prec
c       zc()      = spline parameter array                  dbl prec
c       zd()      = spline parameter array                  dbl prec

c  last update 27/10/93  sbh


      subroutine linear( r  , z  , s  , t  , mp ,
     &                   ra , rb , rc , rd , np ,
     &                   za , zb , zc , zd , dn , theta )
```

```
c  Dimension variables
c  -------------------

       implicit double precision ( a - h , o - z )

       dimension r( 0 : mp ) , z( 0 : mp ) , s( 0 : mp ) ,
     &           t( 0 : mp ) , theta( 0 : mp )

       dimension
     &   ra( 0 : mp ) , rb( 0 : mp ) , rc( 0 : mp ) , rd( 0 : mp ) ,
     &   za( 0 : mp ) , zb( 0 : mp ) , zc( 0 : mp ) , zd( 0 : mp )


c  Set constants
c  -------------

       pi = 4.0d0 * atan( 1.0d0 )


c  Set up dummy cubic spline parameters for ease of portability
c  ------------------------------------------------------------

       do 10 j = 0 , np , 1

           ra( j ) = r( j )
           za( j ) = z( j )
           rc( j ) = 0.0d0
           zc( j ) = 0.0d0
           rd( j ) = 0.0d0
           zd( j ) = 0.0d0

 10    continue


c  Set up orientation array
c  ------------------------

       do 20 j = 1 , np - 1 , 1

           thjmh = atan( ( r( j - 1 ) - r( j ) ) /
     &                   ( z( j ) - z( j - 1 ) ) )

           thjph = atan( ( r( j ) - r( j + 1 ) ) /
     &                   ( z( j + 1 ) - z( j ) ) )
```

```
            theta( j ) = s( j ) / ( s( j ) + s( j + 1 ) ) *
     &                   thjph +
     &                   s( j + 1 ) / ( s( j ) + s( j + 1 ) ) *
     &                   thjmh

 20     continue

        theta( 0 ) = -1.0d0 * pi / 2.0d0
        theta( np ) = pi / 2.0d0

        do 30 j = 1 , np - 1 , 1

            rb( j ) = -1.0d0 * sin( theta( j ) )
            zb( j ) = cos( theta( j ) )

 30     continue

        rb( 0 ) = 1.0d0
        rb( np ) = -1.0d0
        zb( 0 ) = 0.0d0
        zb( np ) = 0.0d0


c  Housekeeping
c  ------------

        return
        end
```

Appendix D

GRAPHIC DISPLAY PROGRAM

```
!!MP

c       ****************************************************************
c       *                                                              *
c       *             MECH 955 - MAJOR ME THESIS  (1992)               *
c       *                 Course code 303 / ME(Hons)                   *
c       *                                                              *
c       *                  STEVEN HARVEY    8421248                    *
c       *                                                              *
c       *                 Supervised by Dr W. K. Soh                   *
c       *                                                              *
c       ****************************************************************
c       *                        Program BUBEXS                        *
c       ****************************************************************
c       *                                                              *
c       * This program displays each bubble on a Macintosh screen     *
c       * as read in from its image coordinate file. By not            *
c       * performing screen erases between files the entire            *
c       * evolution is captured.                                       *
c       *                                                              *
c       * For raw EXPERIMENTAL (x-y) data only.                        *
c       *                                                              *
c       * Compiled using Language Systems FORTRAN 2.1 in MPW 3.2.      *
c       *                                                              *
c       ****************************************************************
```

```
c       Variable list
c       -------------

c       crdfl()  = coordinate filename array                character
c       done     = test condition flag                      logical
c       dum      = dummy variable                           character
c       dummy    = dummy variable                           real
c       InitGraphWindow = graphics primitive                integer
c       maxfil   = maximum no. of files allowed             integer
c       maxnod   = maximum no. of nodes allowed             integer
c       Mouse    = mouse event flag                         logical
c       myWindow = graphics primitive                       integer
c       n        = dummy variable                           integer
c       narea    = dummy area variable                      integer
c       nnodes   = total no. of nodes - 1                   integer
c       ntime    = dummy time variable                      integer
c       numfls   = no. of coordinate files to be read in    integer
c       nxi      = horizontal coord. dummy variable         integer
c       nybolt   = dummy reference point variable           integer
c       nyi      = vertical coord. dummy variable           integer
c       scal     = overall screen scaling factor            dbl prec
c       time()   = time array                               character
c       title    = screen title                             string
c       tmstep   = timestep betwen subsequent coord files   dbl prec
c       xi(0:,)  = horizontal coordinate array              dbl prec
c       xmax     = maximum horizontal coordinate            dbl prec
c       xmin     = minimum horizontal coordinate            dbl prec
c       xscal    = horizontal screen scaling factor         dbl prec
c       xscrn    = horizontal screen coordinate             integer
c       ybolt()  = reference bolt vertical coord. array     dbl prec
c       yi(0:,)  = vertical coordinate array                dbl prec
c       ymax     = maximum vertical coordinate              dbl prec
c       ymin     = minimum vertical coordinate              dbl prec
c       yscal    = vertical screen scaling factor           dbl prec
```

```
c       yscrn    = vertical screen coordinate                integer
c       ybdata   = filename of reference point file          character

c  last update 10/8/93  sbh


        program bubexs


c  Dimension variables
c  -------------------

        parameter( maxnod = 36 , maxfil = 180 )
        integer*4 InitGraphWindow
        integer*4 myWindow
        integer*2 xscrn , yscrn
        character*11 crdfl( maxfil ) , ybdata
        character*40 time( maxfil ) , dum
        logical*4 done , Mouse

        double precision xi( 0 : maxnod , maxfil ) ,
     &                   yi( 0 : maxnod , maxfil ) ,
     &                   xmin , xmax , ymin , ymax , tmstep ,
     &                   xscal , yscal , scal ,
     &                   ybolt( maxfil )

        string*255 title


c  Initialise variables
c  --------------------

        myWindow = InitGraphWindow( 0 , 0 , 0 , 0 )
        xmin = 0.0d0
        xmax = 0.0d0
        ymin = 0.0d0
        ymax = 0.0d0
        done = .false.
        title = 'Evolution of one vapour cavity'
```

```
c  Open data files and read in coordinate data (0 to 18 )
c  ------------------------------------------------------

        call SetPoint( int2( 5 ) , int2( 60 ) )
        call DrawText('Coordinate files being read in')

        open( 1 , file = 'crdfls.inp' )
        rewind( 1 )
        read( 1 , * ) nnodes
        read( 1 , * ) numfls
        read( 1 , * ) dummy
        read( 1 , * ) dummy
        read( 1 , * ) dummy
        read( 1 , * ) dummy
        read( 1 , * ) dummy
        read( 1 , * ) dummy
        read( 1 , * ) dummy
        read( 1 , * ) dummy
        read( 1 , * ) ybdata
        read( 1 , * ) dummy
        read( 1 , * ) dummy


c  Read in filenames
c  -----------------

        do 20 i = 1 , numfls , 1

            read( 1 , 500 ) crdfl( i )

 20     continue

        close( 1 )
```

```
c  Read in the values of ybolt for each coordinate file
c  ------------------------------------------------

        dum = 'time (us) =           '
        open( 1 , file = ybdata )
        read( 1 , 510 )

        do 30 i = 1 , numfls , 1

            read( 1 , 540 ) ntime , narea , nybolt
            ybolt( i ) = dble( nybolt )
            write( fmt = 2000 , unit = time( i ) ) ntime
            time( i ) = 'time (us) = ' // time( i )

 30     continue

        close( 1 )


c  Read in the input coordinates
c  -----------------------------

        do 40 i = 1 , numfls , 1

            open( 1 , file = crdfl( i ) )
            read( 1 , 510 )

            do 50 j = 0 , nnodes , 1

                read( 1 , 520 ) n , nxi, nyi
                xi( j , i ) = dble( nxi )
                yi( j , i ) = dble( nyi )

 50         continue

            close( 1 )

 40     continue
```

```
c  Compute the maximum and minimum coordinates
c  ------------------------------------------

        do 60 i = 1 , numfls , 1

            if ( ybolt( i ) .gt. ymax ) then
                ymax = ybolt( i )
            endif

            do 60 j = 0 , nnodes , 1

                if ( xi( j , i ) .gt. xmax ) then
                    xmax = xi( j , i )
                endif

                if ( xi( j , i ) .lt. xmin ) then
                    xmin = xi( j , i )
                endif

                if ( yi( j , i ) .lt. ymin ) then
                    ymin = yi( j , i )
                endif

 60     continue

        ymax = ymax + 15.0


c  Calculate overall screen scaling factor
c  --------------------------------------

        xscal = abs( xmax - xmin )
        yscal = abs( ymax - ymin )

        if ( xscal .gt.  yscal ) then
            scal = xscal
        else
            scal = yscal
        end if
```

```
c  Initialise output window
c  ------------------------

       call EraseGraphWindow( myWindow )


c  Scale node coordinates to fit screen and plot bubble shapes
c  -----------------------------------------------------------

       do while ( .not. done )

           do 70 i = 1 , numfls , 1

               yscrn = int2((( ybolt( 1 ) - ymin )/scal)
     &                        * 312.0 )
               call SetPoint( int2( 0 ) , yscrn )
               call Draw( int2( 512 ) , yscrn )

               xscrn = int2((( xi( 0 , i ) - xmin )/scal)
     &                        * 312.0 + 50.0 )
               yscrn = int2((( yi( 0 , i ) - ymin )/scal)
     &                        * 312.0 )

               call SetPoint( xscrn , yscrn )

               do 80 j = 1 , nnodes , 1

                   xscrn = int2((( xi( j , i ) - xmin )/scal)
     &                            * 312.0 + 50.0 )
                   yscrn = int2((( yi( j , i ) - ymin )/scal)
     &                            * 312.0 )
                   call Draw( xscrn , yscrn )

80             continue

                   xscrn = int2((( xi( 0 , i ) - xmin )/scal)
     &                            * 312.0 + 50.0 )
                   yscrn = int2((( yi( 0 , i ) - ymin )/scal)
     &                            * 312.0 )

               call Draw( xscrn , yscrn )

70         continue
```

```
c           Time delay to enable screen to be captured
c           ------------------------------------------
            do 100 ii = 1 , 300000 , 1
 100        continue


c           Reset screen and replay if necessary
c           ------------------------------------

            call EraseGraphWindow( myWindow )

            done = .false.

            if ( Mouse() ) then
                done = .true.
            endif

        enddo


c  Housekeeping
c  ------------

        call CloseGraphWindow( myWindow )

 500    format( a11 )
 510    format( / )
 520    format( i4 , t10 , i4 , t18 , i4 )
 540    format( t17 , i5 , t31 , i5 , t98 , i4 )
 2000   format( i8 )

        end
```